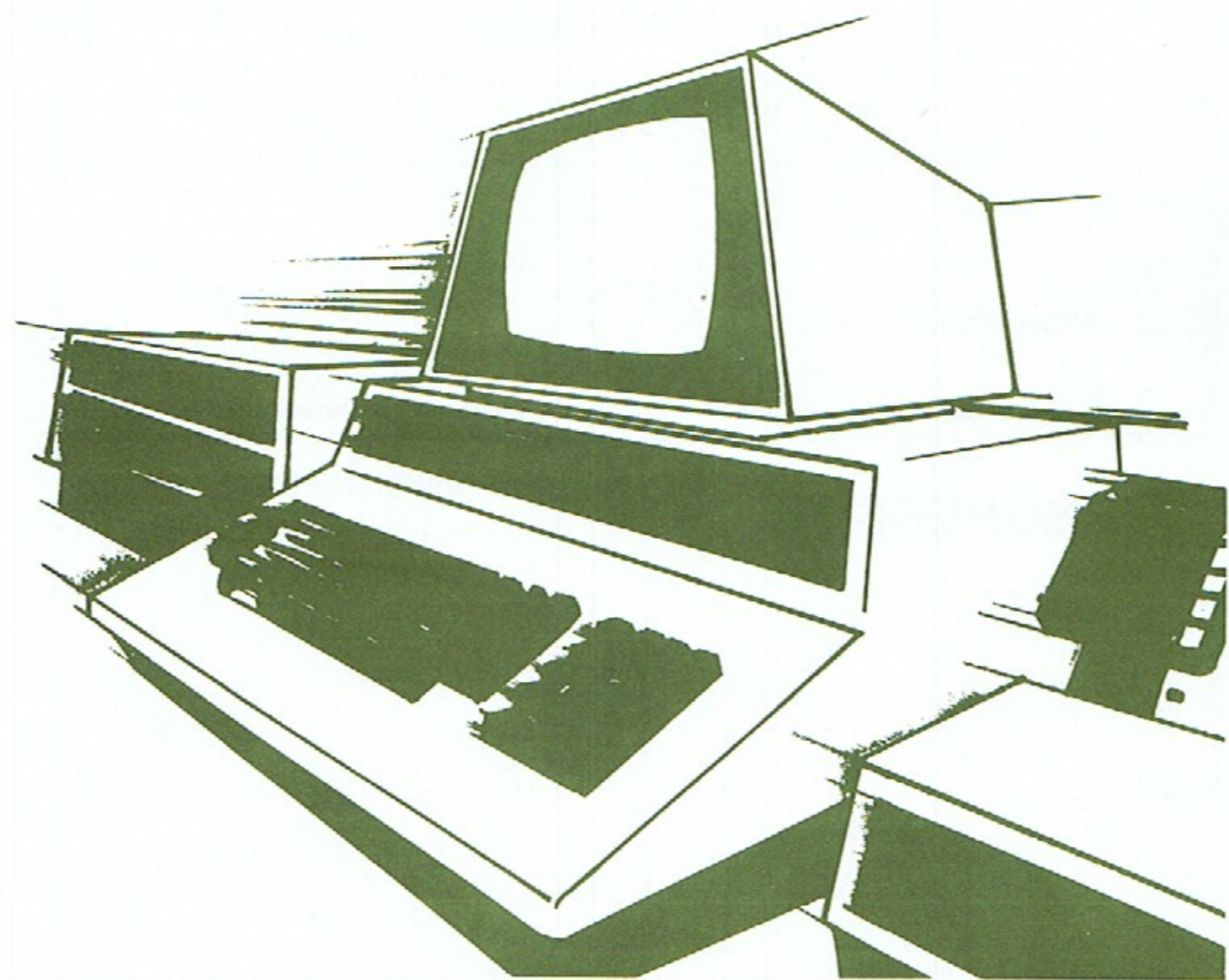


CPUCN

The Official Commodore Pet Users Club Newsletter



Volume 2

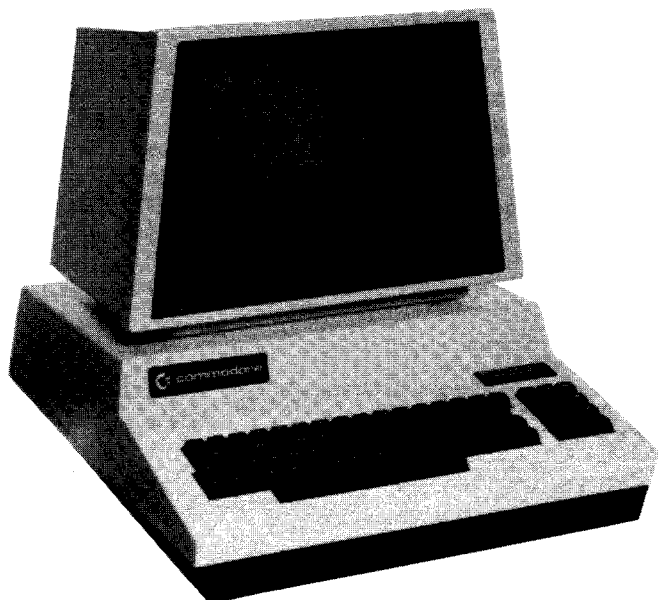
Issue 8

 **commodore**

Contents

Editorial.....	1
Commodore News.....	2
Book Reviews.....	3
Business Users.....	4
Pascal Programming.....	6
BASIC Programming.....	8
Communicator.....	15
Machine Code.....	16
User Club Bits & Pieces.....	26

Editorial



Dave Middleton

The 8000 series - or SuperPET

The long awaited 80 column screen PET is now available from Commodore Commercial Systems dealers around the country. The 3000 series maintain compatibility with the 3000 series for most applications where BASIC is used.

The 8000 series support what is called BASIC 4.0 or PET disk BASIC. 14 extra commands have been added which allow far easier communication with the disk unit. There are also two reserved variables DS and DS\$ which are used to interrogate the disk system for error messages. For instance DLOAD"PROG-NAME" will load 'PROG-NAME' into RAM ready to be RUN.

The disk commands are only recognised by DOS 2.5 which is in the 8050 1 megabyte disk drive but the chips will be available

for retrofit in the near future from your dealer. Note also that a special version of BASIC 4.0 is being produced for the 3000 series PET so that the advanced features will be available to all users.

The other new item for the 8000 series is of course the screen. This has many advanced features apart from being 80 characters wide. There are for instance two mode of operation, text mode where there is small gap between the lines making text easier to read and graphics mode where the lines are closed up as with the 3000 series. A simple command allows easy switching between the two. It is also possible to define a 'window' in which all text is to be printed, leaving the rest of the screen untouched; once again a couple of simple commands allow a window of any size to be formed on the screen. The screen RAM is now 2000 characters (80x25 lines) and this may have an effect on some programs which address screen locations with PEEK/POKE commands.

1	2	3	4	1	2	3	4	5	6	7	8
5	6	7	8	9	10	11	12	13	14	15	16
9	10	11	12								
13	14	15	16								

40 col.

80 col.

Thus every 2nd line on the 40 column screen is now at the right hand side of the 80 column, so displays which use actual screen locations such as POKE33768,160 will look very strange when transferred directly to the 8000 series from the 3000 series.

Another advancement is in the garbage collection routine which could 'hang' the PET for up to 10 minutes while it removed rubbish from string storage. Also it is now possible to use repeat on all keys so there is no need to, for instance, keep stabbing at the cursor control keys to get to a specific point on the screen. Anybody who has tried editing a program will know how frustrating it can be trying to get out of 'quotes' (") mode to use the cursor keys well all you have to do with the 8000 series is press the 'ESC' key and you can continue to edit the line as if nothing had been changed.

The 8000 series has a standard QWERTY keyboard which will be a great relief to a lot of experienced secretaries who must have been greatly perturbed by the 3000 series with the full stop on the numeric keypad.

Remember how any program which used zero page memory in the Old and New ROM 3000 series had to be carefully modified? BASIC 4.0 maintains the same zero page addresses as the New ROMs for the 3000. The only changes being in the screen wrap for cursor control to allow the 40 column PET to have 80 column BASIC lines. These location are superfluous to the 8000 series and are used for, amongst other things, controlling the window.

Commodore News

Andrew Goltz
Market Support Manager

Fast Printer

A new Commodore printer, the 8024, designed to complement the 8032 "SuperPET" and 8050 1 megabyte disk unit is being formally announced to the press on the 16th of September.

The printer which has the standard ASCII 96 character set, operates at 160 characters per second and is ideal for the business-man who requires lots of hard copy, really fast. Typical applications will include accounting and mailing lists. The 8024 will accept paper from 4 to 15 inch widths and will have no difficulty in printing address labels. Incidentally the printer will of course work with the 3000 series PETs as well!

Other specifications include a 9x7 dot matrix, double width characters under software control, tractor feed and the capability for printing one original and up to 4 copies on standard continuous fan fold paper. There will be a full review and a price announcement in the next issue of CPUCN.

The Commodore Management System

Also being announced on the 16th of September is a new range of business programs from Commodore specially designed around the enhanced capabilities of the 8000 series hardware.

Designated the 'The Commodore Management System' the individual packages comprise: 'The Accountant', 'Paymaster' and 'Ozz - The Information Wizard'. Full details will be found in our latest sales brochure: "Commodore News" which you should have received with this edition of CPUCN. Further copies are available from the Commodore Information Centre, 360 Euston Road, London NW1.

Ozz - The incredible

"Commodore News" is worth reading if for no other reason than to obtain further details on 'Ozz' - it is a really flexible data-base package specially written for Commodore by the well respected software house - The Bristol Software Company. Written entirely in Assembler - 'Ozz' can be individually tailored to suit literally hundreds of applications and the tailoring can be performed by somebody with absolutely no programming experience. Alternatively 'Ozz' can be set up for individual application by your nearest Commodore Commercial Systems dealer for a charge, depending on the complexity of the work, of between 150 and 250 pounds. Remember that bespoke packages will cost up to 10 times these prices.

It is worth pointing out that only official Commodore Commercial Systems dealers will be carrying the 8000 Series PETs, 8050 disk units, Commodore Management System programs and Business Software for the 3000 series machines. Also note that this new 8000 series software is due for official release between October and Christmas. Further details and reviews will be given in the next edition of CPUCN.

New packages for the 40 column PETs

40 column PET owners will be please to know that it is Commodores intention to continue to produce AND support the 3000 series machines. Among the new Commodore Software packages being announced on the 16th September is a new accounting system to run on the 3000 series PETs and a superb Stock Control program written by Anagram who were formally in the Commodore Approved scheme.

Comaccounts transfered to Approved Products scheme

The Comaccounts suite of accounting programs have been installed and are successfully in use in many locations around the country, however the complexity of the programs is such that Commodore feel that end users would be better supported if the programs were sold directly by the originating software house, Microcomp and a number of specialised dealers working in close liason with the company. Accordingly the programs will now be marketed directly by Microcomp as a Commodore Approved Product.

Petpacks

Pete Gerrard

As most of you will be aware by now, the introduction of the Arcade series into our PETpack Master Library has been a tremendous success. The first three games in this series (Invaders, Acrobat and 3D Startrek) will be familiar to you from reviews in earlier newsletters (Vol. 2, Issues 3 & 4), and they all continue to have terrific sales

You will have seen from the catalogue with your last newsletter that the Arcade series has been extended by three further titles, namely Breakthrough, Night Drive and Car Race. They too are proving to be hot favourites, so it would seem that there is a demand for high quality, low-cost games software. Low-cost is a theme we have always tried to stress in our games (and other) releases - to play an infinite number of games of Invaders for 7 pounds cannot be bad. To continue this trend, you will also have seen in the catalogue last time a program called 'Galaxy One', a disk based program in the GD series (order number GD1800), for 40 pounds. I say 'a' program - it is worthy of a better mention than that!

Galaxy One in fact consists of 24 separate programs, culled from the original Treasure Trove cassette based series, and featuring all 24 of the games contained on the first six cassettes in that series. So what? It means you can load each game instantly, without having to search through a whole tape if your particular favourite happens to be the last game on the tape. It works out to just 1.66 pounds per game as opposed to 2.50 pounds. You get a smart looking manual instead of a series of simple backing cards. And so on...

The games themselves cover a wide variety of topics, from card games to space games, board games to simulations and infuriating to amusing games that make great use of the PET's graphics. These are some of the earliest games written for the PET, and as such mark an important era in its history. And although they are early programs, that does not mean they are lacking sophistication. Not up to the Arcade series standards perhaps, but then nothing is. It could not be an Arcade game otherwise! Some of these early games, although simple to play, become incredibly addictive after just a few games, and sparked a whole series of ideas in the minds of people who later would incorporate those ideas into other, non-game, programs. So they are not without interest, despite as I say being written in the early days of PET.

Going onto some of the games individually, here are some that deserve special mention, the ones that have become firm favourites over the years. Games like Lunar Landing, Target Pong, Galaxy Games, Wrap Trap, Super X9, Backgammon and so on. All are easy to learn, but it is very difficult to beat the machine and since they are on disk they are all very quickly loaded.

Since this idea has already shown itself to be a success, there will be further editions in this series in the near future. The next one will consist of 24 games, like this one (and for the same price), but this time they will all be games you have not seen before!

And staying with the topic of new releases the next issue will give full details of all the autumn plans for the cassette library. One release to watch out for is an educational disk, containing 30 programs, so keep your eyes open for the next issue.

Thank you for all the programs you have submitted so far - keep them coming!

Book Reviews

Dave Middleton

If your company produces a book please send a copy to the Editor for a review. When you send the book please include price and availability. This is a service that is useful to both the readers of CPUCN and to the author/retailer of the book.

Understanding Your PET/CBM
Vol.1 - Basic Programming
By: D.Schultz and D.Smith

Cost: 15 pounds

Available from: PETSOFT

The book is 240 pages long and is packed with information which the absolute beginner will find invaluable. It would appear that the book is mainly a compilation of the popular TIS workbooks with the bemused lion making explanatory appearances throughout. The text starts off each section with very simple examples and then builds on these slowly to show more complex methods.

The sections cover just about everything needed to get a minimum system (ie. PET + cassette deck) into operation. Unfortunately the amount written about using a disk system with the PET can be held on one page which is a bit of a shame when 46 pages are devoted to the cassette. Possibly the next volume, which is not available yet, will contain disk programming.

If anything the book takes teaching by example too far and the user may become bored plodding through every program but this is something which is an asset rather than hinderance as it means that nothing is assumed by the authors and if the reader wishes to skip a few pages then it is their own fault if they do not understand later. It is this fact which a lot of text books could do to take example from, too much information is far better than too little.

Dave Briggs

PET and the IEEE 488 Bus (GPIB)

By: Eugene Fisher-CW Jensen.

Publishers: McGraw-Hill

Cost: 9.95 pounds

Available from: Audiogenics

If like me, you have often used the PET IEEE Bus and consider that you have a fair idea of how it works then this book could come as quite an eye opener.

With prolific use of tables and timing diagrams the first six chapters give a detailed and concise description of how the PET IEEE Bus operates for the various combinations of commands that are available. Non-standard interfacing and the differences between the PET IEEE and the standard GPIB are also covered in the same lucid style.

A chapter on specialised applications is the only one that goes into more depth and technicalities than the first time user will require.

The book finishes with a chapter on how to find out what is wrong when the Bus is apparently not working correctly and a series of appendices, including a very comprehensive bibliography, add an equally comprehensive listing of devices with GPIB interfaces.

All in all the book gives an excellent insight into the workings of the PET IEEE 488 Bus which will be of value to both first time and more advanced users. It should definitely appear in the library of all serious PET users.

Business Users

Barry Miles

THE NEW BUSINESS USER'S COLUMN

Your editor has commissioned this series of articles especially for the new member of the club who is also a business user, and who wants to know how to get the best out of his machine.

The experienced user and the hobbyist are both well catered for in CPUCN, but it is highly likely that the business user, confronted with the machinery, probably for the first time, will feel the need for some guidance.

As in the field of larger installations, it is very likely that some users will have been persuaded to buy first, and find out what it's all about later!

Thus some of the information will be directed towards the choice of peripheral hardware and of programs.

The series will try to avoid unnecessary jargon as much as possible and try to indicate some of the many pitfalls awaiting the unwary user. Particular emphasis will be given to those areas which receive light treatment in any literature which is likely to fall into the hands of the business user or, more likely, no treatment at all!

To whet your appetite, here are some of the matters which will be examined in forthcoming articles:

How do you set about the task of selection from the bewildering array of peripheral machinery available; which attributes are important, and which are not?

How can you keep your staff happy in the changing environment brought about by computerisation?

Should your auditor be involved and if so, at what stage?

To what extent will your computer cause changes in your procedures and in your thinking about systems?

How should you plan for the future of your installation?

What can be done about machinery breakdowns, and how can you minimise the effects?

How can you ensure that fluctuations in the voltage of your power supply will not cause a loss of vital business data?

How can you obtain most of your computer applications virtually cost-free?

How can you carry out your feasibility study in the cheapest and most effective way possible?

What can be done to ensure the security of

your programs and vital business data?

How can you seek to avoid an increase in the possibility of fraud in your business?

What steps should you take to rationalise data preparation, so as to increase efficiency.

How can you avoid your staff's spending much of their day playing Startrek or Space Invaders; if you can avoid this, should you?

How do you set about assessing the quality of suppliers and designers of programs?

Are there any programs of such universal applicability that you should buy them without even going through the motions of checking them out? If so, is there a variety of these to choose from and how should one set about choosing them?

How do you set about choosing business software anyway?

If these matters are of interest then this column is for you: Watch this space!

Calco Software present the....

Pronto-Pet

....hard/soft reset switch

The Pronto-Pet is a single push-button with double action - either a complete reset or break to the machine code monitor.

HARD RESET clears the Pet and peripherals, equivalent to switching the power off and on

SOFT RESET breaks into the monitor when the Pet has "crashed", so that you can save your program and find the cause of the crash.

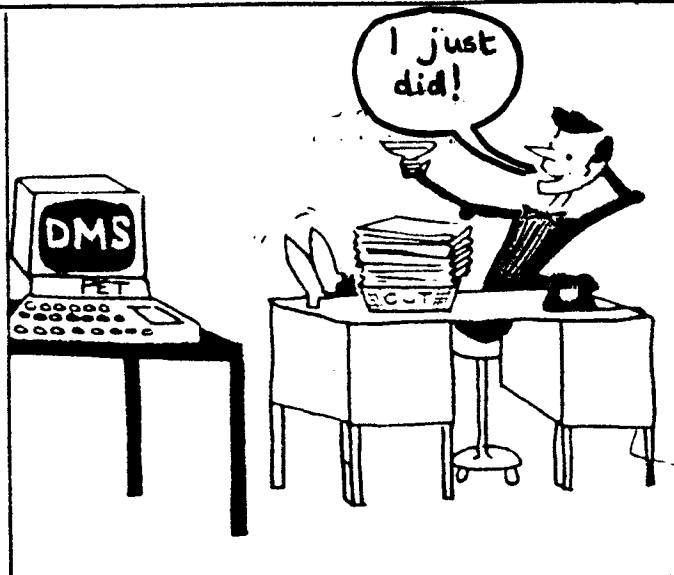
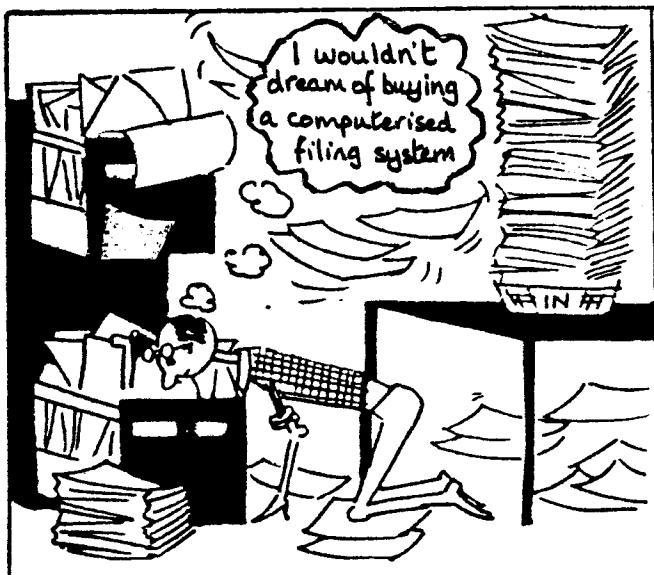
Simple to use, and simple to fit. Open your Pet - connect four push-on pins - press the Pronto-Pet onto base of Pet - close the Pet.

The Pronto-Pet is machined from black anodised aluminium and does not disfigure your Pet. For standard new-Rom models only.

Available ex-stock at only £9.99 + VAT, post paid, by cheque/access/barclaycard. See your nearest dealer or phone 01-546-7256, anytime

Calco Software

Lakeside House, Kingston Hill, Surrey KT2 7QT 01-546-7256



RECORD KEEPING MADE SIMPLE ON THE 32K DISK BASED PET.

- Stores information
- Updates records
- Searches
- Prints and sorts reports
- Prints labels
- Easy to use: no programming knowledge needed

••• DOES ANY CALCULATIONS e.g. VAT, SALES FIGURES, INVENTORY VALUES, PRICE INCREASES, etc, etc, etc.

A unique program; original and best. Over 200 sold in UK alone.

Available from Compsoft, Old Manor Lane, Chilworth, Surrey. Tel. Guildford (0483) 39665 and Geneva (022) 574834 and Commodore in Hong Kong and Canada, and from PET dealers nationwide

EXISTING USERS: PLEASE REGISTER WITH COMPSOFT IMMEDIATELY FOR A COMPLIMENTARY COPY OF OUR NEW VERSION.

Soon available on CP/M.

PAYROLL 'PLUS'

£150 plus VAT

This must be the finest plain paper payroll available for the CBM PET.

It is designed to the Inland Revenue Specifications for Computerised Payroll. It uses plain computer paper throughout and so avoids the need for expensive pre-printing and the annoyance of having to change the paper for specific uses.

Included in its coverage is the following: ALL Tax Codes, ALL NI Codes. Hourly, Weekly and Monthly paid staff — and mixed on the same file disk. 3 rates of overtime which can be entered as amounts or as percentages for hourly staff. 5 Pre-tax adjustments — 2 of which may be pre-set to avoid re-entry each payday. 5 After tax adjustments — again 2 of these may be pre-set. Easy manipulation of employee data — under a security password (which may be changed). Listing for P35. Will handle up to 500 employees on one data disk — and all can be current. Employee deletion without affecting totals.

Four choices of payroll run method: (1) Payslip print-out after each entry. (2) All entries made first, then continuous print run. (3) Immediate payslip print run without entries — if payroll is suitable. (4) Select individual employees.

Payslips are very comprehensive and easy to read and payslips and copies are printed side by side so that the employers copies may be kept in a continuous strip. The extra NI figures required for Contracted-Out employment are printed.

An analysis after the pay run gives Taxable Pay, Employers NI, Deductions and Totals — in other words, the actual cost of the employment and this is in up to 26 separate groups. These are followed by the total Overtime hours for each of the 3 rates, then the full combined totals and a Cash Analysis.

Landsoft Payroll Programs are in use by a considerable number of Accountants and are known for their simplicity of operation and 'User Friendliness'.

HOTEL GB

£350 plus VAT

This fast elegant program is the answer to the hoteliers dreams. It makes the invoicing of guests for their accommodation and services extremely easy. No longer the chore of entering all the accommodation charges every night, the computer does it automatically. At the touch of a few keys a guests account to date can be displayed and the bill printed with a copy for the hotel.

Daily and period totals for 22 service items can be had whenever required. Also grand totals, Total debt to hotel. Items deleted from accounts. Payments in cash. Payments by five different credit cards. Deposits etc.

Hardware and Software will cost little more than half the price of a custom guest billing machine — and the computer gives the ability to do Payroll, Stock Control and General Accounts.



SUPERIOR PROGRAMS FOR THE 32K CBM AND CBM DISK

LANDSLER SOFTWARE 29a Tolworth Park Road, Surbiton, Surrey. 01-399 2476/7

Pascal Programming

PASCAL

The following article was written by Rob Evans and will give an insight into the new Pascal compiler for the PET. A full description of what the compiler is capable of is impossible to give in a few lines so Rob has detailed the differences between Commodore Pascal and that laid down in the 'Pascal user manual and report' by Jensen and Wirth. There is a second article giving details of an actual Pascal program written by Rob based on the game Mastermind, this will be published in CPUCN 3.1

Keith Frewin left University College London with a 1st Class Honours degree in Electronic Engineering and Computer Science and started work with Transam Computers, a year later he produced the first Pascal compiler to run in under 32k of RAM for the Triton micro-computer. Commodore, interested in another powerful language approached Transam and 4 months later the Pascal compiler for the PET made its appearance. There are very few differences between the two versions of the compiler; provision had to be made for a different disk operating system and the slightly lower memory available on the PET but the compiler accesses the PETs floating point package in ROM thus keeping the 9 digit accuracy for high precision arithmetic.

Large portions of the compiler were written in Pascal with machine code being used where speed is of high importance hence Keith only had to change the machine code sections to get the PET version running, thus showing the great portability of Pascal.

Keith is now in the process of converting the compiler to run on the 80 column PET.

Rob Evans

PASCAL REVIEW

Before giving you my review of Pascal I have been asked to say a few words about myself.

Although at present I work for Commodore's software department I am, as it were, on loan from Brunel University for a period of 22 weeks. The reason is I take a computer-science undergraduate sandwich course, which includes 3 industrial training periods of 22 weeks and Commodore kindly employ me for the second of these (incidentally anyone interested in employing a Brunel student in a similar capacity next year can contact me at Commodore now!).

A large amount of my time at Commodore has been spent testing the Pascal system and editing the manual supplied with it - and a most enjoyable job too! However all good things must come to an end and Pascal went onto the market in mid-July. Incidentally, we produced 1,000 copies of Pascal on the first production run, some of which have

gone abroad, as we are confident it will prove very popular.

Anyone interested in name derivation may know that Pascal was named after the French mathematician Blaise Pascal. Who, according to 'Who Did What', invented a calculating machine back in the 17th century and was presumably thus honoured. But if Pascal were an acronym then how about Programming with A Structured Concise Algorithmic Language - any better suggestions?

By the way we shall soon be starting a Pascal users' group, initially as a couple of pages in CPUCN. Anyone interested in contributing anything or in helping to answer the questions from other Pascal users please write to the editor at Commodore Slough.

The hardware required to develop and run Pascal is the CBM Professional computer with 32K RAM and BASIC 2.0, plus a model 3040 floppy disk unit with DOS 1.0.

Review

Firstly and perhaps most important is the cost, which is 120 pounds. For this you get the Pascal development system on diskette plus a comprehensive 100 page manual which comprises 3 chapters; Introduction to Pascal; Beginner's Guide to Pascal; and the Pascal Reference Manual.

At the outset let me say that this version of Pascal contains all the features included in the 'Pascal User Manual and Report' by Jensen and Wirth, as well as many additional features which will be subsequently described.

The Pascal system comprises a number of programs and files which are needed to develop and run your programs. Tackling them in the order you would use them when developing a program the first is the EDITOR (4k). The editor exists to allow Pascal programs to be keyed in and saved under a file name with the greatest convenience to the programmer. The Pascal source program written in pseudo English (i.e. Pascal!), while convenient for you, is not understood by the PET which only understands it's own language - machine code. So the second program provided is the COMPILER (14k) which converts your Pascal source program into a Pascal object program - know as P-code. P-code is an intermediate code between source code and machine code and so requires another program, the INTERPRETER (10k), to read the P-code and execute it.

However, starting at the beginning, Pascal is loaded as any other program would be and from this point on you are in Pascal resident mode, although it is easy to switch in and out of BASIC as you will subsequently see. Resident refers to the Pascal compiler, which is resident in RAM, and is distinct from the disk mode when the compiler is held on disk (until required) - disk mode allows an extra 24k for your source program, 28k in all. - In resident mode you are operating in a similar mode to

which you might be used to with BASIC. Each statement is given a line number and when all the program is keyed in, it is executed by simply typing RUN. All Pascal statements are available in resident mode except disk commands and obviously there is a restriction as to the size of the source program. So, summing up, resident mode facilitates the fast development of smaller, simpler Pascal programs and primarily is envisaged as an aid to becoming proficient in Pascal very quickly. When you write your programs for real then you will use disk mode.

In disk mode, you can write larger programs because the compiler resides on disk. Also, in this mode ALL features of the Pascal system are available. Disk mode operates in a similar fashion as when developing programs on a mini or main-frame computer. When a Pascal source program is created via the editor, instead of immediately executing, it is saved as a file on disk to be subsequently compiled and stored as an object or P-code file. Upon execution of the COMPILE COMMAND, the compiler is read into RAM and the source file read from disk, compiled and written back to disk as a new (object) file under the same name as the source, but with the suffix '.obj'. To execute this program the EXECUTE COMMAND reads the interpreter into RAM and the interpreter reads the program object file from disk and executes it.

Common to both modes of operation is the EDITOR - the program which allows you to enter Pascal source statements. It is based on the BASIC editor but with the following additional features:-

1)Automatic line number generation after the first line is entered.

2)The ability to specify either upper or lower case display. On a personal note I think lower case is infinitely more presentable and in fact you are automatically in lower case when you load Pascal.

3)BASIC may be entered any time simply by typing BASIC and then you may revert back by typing PASCAL.

4)BREAK allows you to enter the machine code monitor.

5)DISK allows you to enter disk mode from resident and RESIDENT switches you back again.

6)Renumbering is achieved with the NUMBER command.

7)Very useful extras are the FIND, CHANGE, and DELETE commands, all of which may be specified within a range.

8)To run programs in RESIDENT MODE use R or RUN, L, or P, where L lists the program on the screen and P on the printer.

9)In DISK MODE file transfer between PET and disk is achieved with the commands GET and PUT.

10)To compile and execute a program in DISK MODE, COMP and EX are available.

11)The HEX and DECIMAL commands are very useful giving instant conversion between the two bases.

12)Finally the LINK command creates one large object module from independently compiled modules as described below.

In addition to the standard features of Pascal there are a number of extras described below, included for the programmer's convenience. These are:-

1)The use of HEXADECIMAL CONSTANTS.

2)PEEK and POKE.

3)A function, called 'ORIGIN', to allow pointers to be set to a physical memory location. For those new to Pascal a pointer is a variable which points to the address of a variable rather than holding the actual value. The use of which is beyond the scope of a review!

4)A function, called VDU, whose arguments accept screen row and column numbers and the character to be displayed thereat. This function is very useful for displaying patterns on the screen!

5)HEXADECIMAL INPUT/OUTPUT is allowed.

6)There are 6 bit-manipulation functions. For example, ANDB accepts 2 arguments and where corresponding bits are set outputs '1' else '0'.

7)You may specify whether input/output errors - for example a character is keyed-in instead of an integer - will cause the system to abort with a suitable error message or simply allow the program to continue.

8)The STOP key may be enabled/disabled as required.

9)A RANDOM NUMBER GENERATOR produces a random number in the range 0-255 and can be manipulated to generate a number in the range 0 to MAXINT.

10)An underscore character can be used to present variable names more neatly (by the way variable names can be up to 16 characters!) for example, NEW_COUNT.

11)The PET internal clock may be both set and read.

12)String variables may be input.
This is extremely useful for
inputting file names.

13) And finally program CHAINING is allowed, as described below.

Anyone concerned about program size restrictions will be pleased to know that careful program design using either CHAINING or LINKING (or both) will in most cases solve this problem.

LINKING allows independently compiled modules to be combined and approximately 3,000 source lines may be thus linked in this fashion. A linked program contains a main-logic module and a number of procedure modules. Each module is compiled independently. Procedures may be called by the main-logic or other procedures. When a procedure is called within a program, program-control passes to it and, on exit from the procedure, returns to the statement immediately after the procedure call. There are many benefits to be had designing programs modularly. For example, the development of each module is isolated from the rest allowing concentration on logically separate sections of the program. Variables common to ALL modules may be specified by declaring the same list of variables at the front of each module. And this is easily achieved by creating a file of variables and using the FILE APPEND command to append the variables at the front of each module. This is extremely neat in practice because if a variable changes then it is simply changed once in the file of variables. Also, debugging is made easier and a recent method I adopted was to display the module name upon entry. Program bugs are then isolated to one module which may be efficiently amended, compiled and relinked with the remaining modules. I would expect most decent-sized programs to be created using linking, which I hope you will find a very effective method after an initial period of familiarisation.

CHAINING allows you to leave one program and enter another from within your Pascal program. Also, if the variables are specified as being the same in both programs (or however many are chained) then the variable's value will be maintained from one program to the next. Using chaining a suite of programs may be run without operator intervention.

Too numerous to list are all the arithmetic and conversion functions but they include the following; sine; natural logarithm and odd (which returns 'true' if the argument is odd and vice versa).

Finally I do not pretend that such a review could pre-empt all possible questions you may have about Pascal, so please do not hesitate to write to the Pascal user group at Commodore Slough should you require further information.

Basic Programming

LIST in Lower Case

Robert Oei, a PET user in Mississauga, has discovered a sequence that will cause PET printers to LIST in upper/lower case rather than graphics and upper case...

The method to accomplish this is actually quite simple:

- ```
1. POKE 59468, 14
2. Enter the following line in immediate
mode:
```

```
OPEN1,4,1:OPEN2,4,2:PRINT#2,"9":FORA=1TO
2:PRINT#1,A;A;A;A;A;A:NEXT:CLOSE1:CLOSE2
```

After pressing RETURN, the printer will print six "1"s and then six "2"s on the next line. If a file is then opened to the printer and the "cmd" command is given, the printer will respond with the word "ready." in lower case. Any program listing performed from then on will appear in upper/lower case. Power down and up to get graphics back.

I regret to say that I don't know what caused this phenomena to occur, except that it works. I would be most interested if you or any PET user can provide a logical reason as to why?... Robert Oei

This was tested on 2022/23 printers and found to work on some but not all.

Paul Barnes  
Deseronto,  
Ontario

## RESTORE DATA Line Program

RESTORE DATA LINE POINTER  
Executing the RESTORE command causes the next READ to occur at the very first DATA element in your program. This subroutine can be used to RESTORE the DATA line pointer at a line other than the first.

It doesn't matter if you don't give the number of the line that has the "DATA" keyword in it that you want to start at, as long as it is past previous DATA statements so that the next data to be read will be the one desired.

PROGRAM NAME: RESTORE NEWROM

```

4 REM*****
5 REM*** RESTORE DATA LINE PRG ***
6 REM*** BY PAUL BARNES ***
7 REM*** DESERONTO, ONTARIO ***
8 REM*** NEW ROM VERSION ***
9 REM*****
10 DATA 166,60,134,17,166,61
15 DATA 134,18,32,44,197,144
20 DATA 11,166,92,142,132,3
25 DATA 166,93,142,133,3,96
30 DATA 162,0,142,132,3,162
35 DATA 0,42,133,3,96

```

```

40 FOR F=826 TO 860: READS: POKEF,S:
NEXT
50 DATA"GOOD-BYE!"
60 DATA"ANYBODY HOME?"
70 J=26545*10: FOR D=1 TO 100: DATA"MAY
BE!"
80 NEXT: DATA"HI!"
100 DATA"GO HOME!"
110 DATA"GO DIRECTLY TO JAIL!"
120 DATA"DO NOT PASS GO!"
130 DATA"DO NOT COLLECT 200 POUNDS!!!"
200 GOSUB1000
210 FOR T=1 TO 3: READ A$: PRINTA$
230 NEXT: PRINT: GOTO200
998 REM***SUBROUTINE TO RESTORE DATA**
999 REM*** AT A CERTAIN LINE NUMBER ***

1000 INPUT"RESTORE TO LINE":A
1010 H=INT(A/256): L=A-H*256
1020 REM POKE CURRENT DATA LINE POINTER

1030 POKE60,L: POKE61,H
1040 SYS826
1050 L=PEEK(900):H=PEEK(901)
1060 IF L=0 AND H=0 THENPRINT"LINE NOT
FOUND": GOTO1000
1070 REM POKE MEMORY ADDRESS OF DATA LI
NE
1080 A=H*256+L-1: H=INT(A/256): L=A-H*2
56
1090 POKE62,L: POKE63,H
1100 RETURN

```

PROGRAM NAME: RESTORE OLDROM

```

4 REM*****
5 REM*** RESTORE DATA LINE PRG ***
6 REM*** BY PAUL BARNES ***
7 REM*** DESERONTO, ONTARIO ***
8 REM*** OLD ROM VERSION ***
9 REM*****
10 DATA 166,142,134,8,166,143
15 DATA 134,9,32,34,197,144
20 DATA 11,166,174,142,132,3
25 DATA 166,175,142,133,3,96
30 DATA 162,0,142,132,3,162
35 DATA 0,42,133,3,96
40 FOR F=826 TO 860: READS: POKEF,S:
NEXT
100 REM=====
110 REM=LINES 50 TO 230 ARE THE SAME=
120 REM=====
998 REM***SUBROUTINE TO RESTORE DATA***
999 REM*** AT A CERTAIN LINE NUMBER ***

1000 INPUT"RESTORE TO LINE":A
1010 H=INT(A/256): L=A-H*256
1020 REM POKE CURRENT DATA LINE POINTER

1030 POKE142,L: POKE143,H
1040 SYS826
1050 L=PEEK(900):H=PEEK(901)
1060 IF L=0 AND H=0 THENPRINT"LINE NOT
FOUND": GOTO1000
1070 REM POKE MEMORY ADDRESS OF DATA LI
NE
1080 A=H*256+L-1: H=INT(A/256): L=A-H*2
56
1090 POKE144,L: POKE145,H
1100 RETURN

```

Jim Butterfield  
Toronto

## CROSS - REFERENCE

One of the handy things about the 2040 disk system is that it allows you to read programs - or write them, for that matter - as if they were data files.

The possibilities are endless: you can analyze or cross-reference programs; renumber them; repack them into minimum number of lines deleting spaces, comments, etc.; or even create a program-writing program that is tailor-made for a particular job.

There are two types of cross-reference normally needed for a BASIC program. It's written in BASIC: that means that it won't run too fast (all those GET statements) but you can read what it's doing fairly easily.

There are two types of cross-references normally needed for a BASIC program. One is the variable cross-reference: where do I use B\$? The other is a line-number cross-reference: when do I go to line 360? CROSS-REF does either. An example of both types is shown - the program in this case did the cross-references of itself.

## CROSS REFERENCE - PROGRAM CROSS-REF

|     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|
| 190 | 470 | 490 |     |     |     |
| 210 | 210 |     |     |     |     |
| 240 | 190 |     |     |     |     |
| 300 | 470 | 500 |     |     |     |
| 320 | 320 |     |     |     |     |
| 360 | 326 | 340 |     |     |     |
| 370 | 330 |     |     |     |     |
| 380 | 310 |     |     |     |     |
| 420 | 380 |     |     |     |     |
| 460 | 440 |     |     |     |     |
| 470 | 390 | 400 | 410 | 430 | 450 |
| 480 | 470 |     |     |     |     |
| 490 | 510 | 520 |     |     |     |
| 530 | 240 |     |     |     |     |
| 580 | 565 |     |     |     |     |
| 600 | 580 |     |     |     |     |
| 620 | 600 |     |     |     |     |

PROGRAM NAME: CROSS-REF

```

100 DIM A$(15),B$(3),X$(500),C(255)
110 PRINT"CROSS-REF JIM BUTTERFIELD"

120 Q$=CHR$(34):S$=" " :B$(1)=Q$:B$(
3)=CHR$(58)
130 INPUT"VARIABLES OR LINES":Z$:C2=5:
IFASC(Z$)=76THENC2=6
140 FORJ=1TO255:C(J)=4:NEXTJ:FORJ=48TO5
7:C(J)=6:NEXTJ
150 IFC2=5THENFORJ=65TO90:C(J)=5:NEXTJ:
FORJ=36TO38:C(J)=7:NEXTJ:C(40)=8
160 C(34)=1:C(143)=2:C(131)=3
170 INPUT"PROGRAM NAME":P$:OPEN1,8,3,"0
":"+P$+","P,R"
180 GET#1,A$,B$:IFASC(B$)<>4THENCLOSE1:
STOP
190 IFB=0GOTO240

```

```

200 PRINTL$:K=X:FORJ=BTO1STEP-1:PRINT
 "A$(J):X=A$(J)
206 X=X+L$
210 IFX$(K)=X$THENX$(K+J)=X$(K):K=K-1:
 GOTO210
220 X$(K+J)=X$:NEXTJ:X=X+B:PRINT:B=0
230 REM: GET NEXT LINE, TEST END
240 GET#1,A$,B$:IFLEN(A$)+LEN(B$)=0
 GOTO530
250 REM GET LINE NUMBER
260 GET#1,A$:L=LEN(A$):IFL=1THENL=ASC(A
 $)
270 GET#1,A$:A=LEN(A$):IFA=1THENA=ASC(A
 $)
280 C=C2:C1=-1:L=A*256+L:L$=STR$(L):IF
 LEN(L$)<6THENL$=LEFT$(S$,6-LEN(L$))
 +L$
290 REM GET BASIC STUFF
300 GET#1,A$:A=LEN(A$):IFA=1THENA=ASC(A
 $)
310 C9=C(A):IFC9>C1GOTO380
320 IFC2=6ANDLEN(M$)<5THENM$=" "+M$:
 GOTO320
326 K=0:IFB=0GOTO360
330 FORJ=1TOB:IFA$(J)=M$GOTO370
340 IFA$(J)<M$THENNEXTJ:K=B:GOTO360
350 FORK=BTOJSTEP-1:A$(K+1)=A$(K):NEXTK

360 B=B+1:A$(K+1)=M$
370 C=C2:C1=-1:M$=""
380 IFC2=5GOTO420
390 IFA=137ORA=138ORA=141ORA=167THENC=6
 :GOTO470
400 IFA=44ORA=32GOTO470
410 IFC9<6THENC=9:GOTO470
420 IFC9=CTHENC=-1:C1=4
430 IFC>6GOTO470
440 IFC<0ANDC9>C1ANDC9>6THENC1=C9:GOTO4
 60
450 IFC2=5THENIFLEN(M$)>20RC>0GOTO470
460 M$=M$+A$
470 ONC9+1GOTO190,480,480,480:GOTO300
480 B$=B$(C9):C$=""
490 GET#1,A$:IFA$=""GOTO190
500 IFA$=B$GOTO300
510 IFA$<Q$GOTO490
520 A$=B$:B$=C$:C$=A$:GOTO490
530 CLOSE1:INPUT"PRINTER":Z$
540 C=3:Z=6:IFASC(Z$)=89THENC=4:Z=12
550 OPEN2,C:PRINT#2:PRINT#2,"CROSS REFE
 RENCE - PROGRAM ":P$
560 X$="" :FORJ=1TOX:A$=X$(J)
565 IFC2=6THENK=6:GOTO580
570 FORK=1TOLEN(A$):IFMID$(A$,K,1)<>" "
 THENNEXTK:STOP
580 B$=LEFT$(A$,K-1):C$=MID$(A$,K,1):
 IFX$=B$GOTO600
590 PRINT#2:Y=0:X$=B$:PRINT#2,X$;LEFT$(
 S$,5-LEN(X$)):
600 Y=Y+1:IFY<ZGOTO620
610 Y=1:PRINT#2:PRINT#2,S$;
620 PRINT#2,LEFT$(S$,6-LEN(C$)):C$;
630 NEXTJ:PRINT#2:CLOSE2

```

#### Reading a BASIC Program as a File

To read a BASIC program, you must OPEN it as a file, using type P for PRG rather than S for SEQ. Line 170 of CROSS-REF does this.

If you read a zero character from the program (that's CHR\$(0), not ASCII zero which has a binary value of 48), the GET# command gives you a small problem: it will give you a null string instead of the CHR\$(0) you might normally expect. You

need to watch this condition and correct it where necessary: you'll see this type of coding in lines 260, 270 and 300.

The first thing to do when you OPEN the file is to get the first two bytes. These represent the program start address, and should be CHR\$(1) and CHR\$(4) for a normal BASIC program starting at hexadecimal 0401 (see line 180).

Now you're ready to start work on a line of BASIC. The first two bytes are the forward chain. If they are both zero (null string) we have reached the end of the BASIC program; otherwise, we don't need them for this job (see line 240).

Continuing on the BASIC line: the next pair of bytes represent the line number, coded in binary. We're likely to need this, so we calculate it as L (lines 260 and 280) and also create it's string equivalent, L\$. We take an extra moment to right-justify the string by putting spaces at the front so that it will sort into proper numeric order.

From this point on we are looking at the text of the BASIC line until we reach a zero which flags end-of-line. At that time we go back and grab the next line.

#### Detailed Syntax Analysis

When digging out variables or line numbers, we have several jobs to do. As we look through the BASIC text, we must find out where the variable or line number starts. For a variable, that's an alphabetic character; for a line number, it's the preceeding keyword GOTO, GOSUB, THEN or RUN followed by an ASCII numeric.

Once we've "aquired" the variable or line number, we must pick up its following characters and tack them on. For line numbers it's strictly numeric digits. For variables, things are more complex. Both alphabetic and numeric digits are allowed, but we should throw away all after the first two since GRUMP and GROAN are the same variable (GR) in PET BASIC. We must also pick up a type identifier - % for integer variables or \$ for strings - if present. Finally, we have to spot the left bracket that tells us we have an array variable.

To help us do this rather complex job, we construct a character type table. Each entry in the table represents an ASCII character, and classifies it according to its type. Numeric characters are type 6. If we're looking for variables, alphabetic characters are type 5, identifiers are type 7, and the left bracket is type 8.

To help us in scanning the BASIC line, we define the end-of-line character as type 0; the quotation mark as type 2; the REM token as type 3; and the DATA token as type 4.

Every time we get a new character from BASIC, we get its type from table C as

variable C9. If we're looking for a new variable or line number, we see if it matches C - alphabetic for variables, numeric for line numbers. Once we find the new item, we kick C out of range and start searching based on the value of C1. This mechanism means that we can search for a variable starting with an alphabetic, and then allow the variable to continue with alphabets, numerics or whatever.

To summarize variables in this area: A is the identity of the character we have obtained from the BASIC program, and C9 is its type. If we're searching, C is the type we are looking for; otherwise it's kicked out of range, to -1 or 9. C1 tells us we're collecting characters and what type we're allowed to collect. C2 is out variables/line numbers flag; it tells us what we're looking for. M\$ is the string we've assembled.

The routine from 480 to 520 scans ahead to skip over strings in quotes and DATA and REM statements.

### Collecting the Results

For each line of the BASIC program we are analyzing, we collect and sort any items we find, eliminating duplicates. They are staged in array A\$ in lines 320 to 370.

When we are ready to start a new line, we add this table to our main results table, array X\$, in lines 200 to 220. To save sorting time, we merge these pre-sorted values into the main table. At this point, our data has the line number stuck on the end; this way, we're handling two values

within a single array.

Because the merging of the two tables must start at the top so that we can make room for the new items, the items are handled in reverse alphabetic order. We print this to the screen so that you can watch things working. At BASIC speed, this program can take quite a while to run; it's nice to confirm that the computer is doing something during this period.

### Final Output

We finish the job starting at line 530. It's mostly a question of breaking the stuck-together strings apart again and then checking to see if we need to start a new line.

### Do Your Own Thing

The size of array X\$ determines how large a program you can handle. The given value of 500 is about right for 16K machines; with 32K you can raise it to 1500 or so.

If you're squeezed for space, change array C to an integer array C%. As you can see from the cross reference listing, you'll need to change lines 100, 140, 150, 160 and 310 - see how handy the program is ?

As mentioned before, run time is slow. A machine language version - or even a BASIC program with machine language inserts - would speed things up dramatically.

NOTE: Some ASCII printers may give double spaced output. If this is a problem the PRINT#2 statements in 590 and 610 should be changed to PRINT#2,CHR\$(13);.

### CROSS REFERENCE - PROGRAM CROSS-REF

|      |     |     |     |     |     |     |     |     |     |     |     |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| A    | 270 | 280 | 300 | 310 | 390 | 400 |     |     |     |     |     |
| A\$  | 180 | 240 | 260 | 270 | 300 | 460 | 490 | 500 | 510 | 520 | 560 |
|      | 570 | 580 |     |     |     |     |     |     |     |     |     |
| A\$( | 100 | 200 | 330 | 340 | 350 | 360 |     |     |     |     |     |
| B    | 190 | 200 | 220 | 326 | 330 | 340 | 350 | 360 |     |     |     |
| B\$  | 180 | 240 | 480 | 500 | 520 | 580 | 590 |     |     |     |     |
| B\$( | 100 | 120 | 480 |     |     |     |     |     |     |     |     |
| C    | 280 | 370 | 390 | 410 | 420 | 430 | 440 | 450 | 540 | 550 |     |
| C\$  | 480 | 520 | 580 | 620 |     |     |     |     |     |     |     |
| C(   | 100 | 140 | 150 | 160 | 310 |     |     |     |     |     |     |
| C1   | 280 | 310 | 370 | 420 | 440 |     |     |     |     |     |     |
| C2   | 130 | 150 | 280 | 320 | 370 | 380 | 450 | 565 |     |     |     |
| C9   | 310 | 410 | 420 | 440 | 470 | 480 |     |     |     |     |     |
| J    | 140 | 150 | 200 | 210 | 220 | 330 | 340 | 350 | 560 | 630 |     |
| K    | 200 | 210 | 220 | 326 | 340 | 350 | 360 | 565 | 570 | 580 |     |
| L    | 260 | 280 |     |     |     |     |     |     |     |     |     |
| L\$  | 200 | 206 | 280 |     |     |     |     |     |     |     |     |
| M\$  | 320 | 330 | 340 | 360 | 370 | 450 | 460 |     |     |     |     |
| P\$  | 170 | 550 |     |     |     |     |     |     |     |     |     |
| Q\$  | 120 | 510 |     |     |     |     |     |     |     |     |     |
| S\$  | 120 | 280 | 590 | 610 | 620 |     |     |     |     |     |     |
| X    | 200 | 220 | 560 |     |     |     |     |     |     |     |     |
| X\$  | 200 | 206 | 210 | 220 | 560 | 580 | 590 |     |     |     |     |
| X\$( | 100 | 210 | 220 | 560 |     |     |     |     |     |     |     |
| Y    | 590 | 600 | 610 |     |     |     |     |     |     |     |     |
| Z    | 540 | 600 |     |     |     |     |     |     |     |     |     |
| Z\$  | 130 | 530 | 540 |     |     |     |     |     |     |     |     |

Printer Fix

There is a small bug in the printer operating system in that it will not respond to a second command sent immediately to the same secondary address. The following program illustrates the problem:-

PROGRAM NAME: PF1

```
100 OPEN4,4: OPEN6,4,6
110 FOR I=1 TO 3
120 PRINT#6,CHR$(24)
130 PRINT#6,CHR$(16)
140 PRINT#4,"██████████"
150 PRINT#4,"██████████"
160 NEXT I
170 CLOSE6
180 CLOSE4
READY.
```

```
██████████
██████████
██████████
██████████
██████████
```

Notice all the lines are the same distance apart. Line 130 is telling the printer to close the gap between the lines up but it is being ignored. This can be corrected by adding a PRINT to a secondary address of 0. This is achieved by line 125 which sends a carriage return without a line feed, the semi-colon stops the 'end of print' line feed being sent.

PROGRAM NAME: PF2

```
100 OPEN4,4: OPEN6,4,6
110 FOR I=1 TO 3
120 PRINT#6,CHR$(24)
125 PRINT#4,CHR$(141);
130 PRINT#6,CHR$(16)
140 PRINT#4,"██████████"
150 PRINT#4,"██████████"
160 NEXT I
163 PRINT#6,CHR$(24)
168 PRINT#4,CHR$(141);
170 CLOSE6
180 CLOSE4
READY.
```

```
██████████
██████████
██████████
```

Lines 163 and 168 are required to set the printer back to standard line feed of 6 lines/inch, RUN does not cause the line feed to be reset to standard.

Program merge from disk

















The following program very easily allows programs to be appended or merged from a disk file in source code not the saved image form (ie. as it would look when LISTed rather than as it is stored as tokens). This allows programs from other systems to be used without manually typing in the program again. Various other programs already exist to perform this function but they rely on fooling the PET into accepting input from a source other than the keyboard. This program simply opens the required file as normal, inputs and displays a line on the screen and then forces the system to accept that line by pushing carriage returns into the keyboard buffer and ENDing. An immediate mode GOTO is also on the screen causing a return to the program when the line has been accepted (more than one GOTO is displayed in order to catch any errors from READY etc.) However the system effectively closes all files when any new line is input, by forcing location 174 to 1 the file is now OPEN again and the process is repeated until the End of File marker is reached. When this point is reached the program deletes itself from the PET and then lists the current contents.


PROGRAM NAME: MERGE

```
63992 INPUT"FILENAME";N$: N$="0:"+N$:
OPEN1,8,2,N$+"S,R": POKE0,0
63993 PRINT"██████";
63994 POKE174,1: IF PEEK(0) THENCLOSE1:
GOTO63998
63995 GET#1,A$: POKE0,ST: PRINTA$: IF
A$<>CHR$(13) GOTO63994
63996 PRINT"GOTO63993": POKE158,5: FOR
I=0 TO 5: POKE623+I,13: NEXT
63997 PRINT"██████GOTO63993": PRINT"███";
END
63998 PRINT"██████": FOR I=0 TO 7: PRINT63
992+I: NEXT: POKE158,10
63999 FOR I=0 TO 9: POKE623+I,13:NEXT:
PRINT"LIST": PRINT"███": END
```

















## CHARACTER CODES



OFF RVS CHR\$ HEX

|                                                                                     |          |            |           |   |
|-------------------------------------------------------------------------------------|----------|------------|-----------|---|
|    | 64<br>0  | 192<br>128 | 192<br>64 | 0 |
|    | 65<br>1  | 193<br>129 | 193<br>65 | 1 |
|    | 66<br>2  | 194<br>130 | 194<br>66 | 2 |
|    | 67<br>3  | 195<br>131 | 195<br>67 | 3 |
|   | 68<br>4  | 196<br>132 | 196<br>68 | 4 |
|  | 69<br>5  | 197<br>133 | 197<br>69 | 5 |
|  | 70<br>6  | 198<br>134 | 198<br>70 | 6 |
|  | 71<br>7  | 199<br>135 | 199<br>71 | 7 |
|  | 72<br>8  | 200<br>136 | 200<br>72 | 8 |
|  | 73<br>9  | 201<br>137 | 201<br>73 | 9 |
|  | 74<br>10 | 202<br>138 | 202<br>74 | A |
|  | 75<br>11 | 203<br>139 | 203<br>75 | B |
|  | 76<br>12 | 204<br>140 | 204<br>76 | C |
|  | 77<br>13 | 205<br>141 | 205<br>77 | D |
|  | 78<br>14 | 206<br>142 | 206<br>78 | E |
|  | 79<br>15 | 207<br>143 | 207<br>79 | F |

















|                                                                                     |           |   |
|-------------------------------------------------------------------------------------|-----------|---|
|  | 141<br>13 | D |
|-------------------------------------------------------------------------------------|-----------|---|



OFF RVS CHR\$ HEX

|                                                                                     |          |            |           |    |
|-------------------------------------------------------------------------------------|----------|------------|-----------|----|
|    | 80<br>16 | 208<br>144 | 208<br>80 | 10 |
|    | 81<br>17 | 209<br>145 | 209<br>81 | 11 |
|    | 82<br>18 | 210<br>146 | 210<br>82 | 12 |
|    | 83<br>19 | 211<br>147 | 211<br>83 | 13 |
|   | 84<br>20 | 212<br>148 | 212<br>84 | 14 |
|  | 85<br>21 | 213<br>149 | 213<br>85 | 15 |
|  | 86<br>22 | 214<br>150 | 214<br>86 | 16 |
|  | 87<br>23 | 215<br>151 | 215<br>87 | 17 |
|  | 88<br>24 | 216<br>152 | 216<br>88 | 18 |
|  | 89<br>25 | 217<br>153 | 217<br>89 | 19 |
|  | 90<br>26 | 218<br>154 | 218<br>90 | 1A |
|  | 91<br>27 | 219<br>155 | 219<br>91 | 1B |
|  | 92<br>28 | 220<br>156 | 220<br>92 | 1C |
|  | 93<br>29 | 221<br>157 | 221<br>93 | 1D |
|  | 94<br>30 | 222<br>158 | 222<br>94 | 1E |
|  | 95<br>31 | 223<br>159 | 223<br>95 | 1F |

















|                                                                                     |           |    |
|-------------------------------------------------------------------------------------|-----------|----|
|  | 147<br>19 | 13 |
|  | 146<br>18 | 12 |



OFF RVS CHR\$ HEX


|                                                                                     |           |            |           |    |
|-------------------------------------------------------------------------------------|-----------|------------|-----------|----|
|    | 96<br>32  | 224<br>160 | 160<br>32 | 20 |
|    | 97<br>33  | 225<br>161 | 161<br>33 | 21 |
|    | 98<br>34  | 226<br>162 | 162<br>34 | 22 |
|    | 99<br>35  | 227<br>163 | 163<br>35 | 23 |
|   | 100<br>36 | 228<br>164 | 164<br>36 | 24 |
|  | 101<br>37 | 229<br>165 | 165<br>37 | 25 |
|  | 102<br>38 | 230<br>166 | 166<br>38 | 26 |
|  | 103<br>39 | 231<br>167 | 167<br>39 | 27 |
|  | 104<br>40 | 232<br>168 | 168<br>40 | 28 |
|  | 105<br>41 | 233<br>169 | 169<br>41 | 29 |
|  | 106<br>42 | 234<br>170 | 170<br>42 | 2A |
|  | 107<br>43 | 235<br>171 | 171<br>43 | 2B |
|  | 108<br>44 | 236<br>172 | 172<br>44 | 2C |
|  | 109<br>45 | 237<br>173 | 173<br>45 | 2D |
|  | 110<br>46 | 238<br>174 | 174<br>46 | 2E |
|  | 111<br>47 | 239<br>175 | 175<br>47 | 2F |


|                                                                                     |           |    |
|-------------------------------------------------------------------------------------|-----------|----|
|  | 145<br>17 | 11 |
|  | 157<br>29 | 10 |


OFF RVS CHR\$ HEX


|                                                                                       |           |            |           |    |
|---------------------------------------------------------------------------------------|-----------|------------|-----------|----|
|    | 112<br>48 | 240<br>176 | 176<br>48 | 30 |
|    | 113<br>49 | 241<br>177 | 177<br>49 | 31 |
|    | 114<br>50 | 242<br>178 | 178<br>50 | 32 |
|    | 115<br>51 | 243<br>179 | 179<br>51 | 33 |
|   | 116<br>52 | 244<br>180 | 180<br>52 | 34 |
|  | 117<br>53 | 245<br>181 | 181<br>53 | 35 |
|  | 118<br>54 | 246<br>182 | 182<br>54 | 36 |
|  | 119<br>55 | 247<br>183 | 183<br>55 | 37 |
|  | 120<br>56 | 248<br>184 | 184<br>56 | 38 |
|  | 121<br>57 | 249<br>185 | 185<br>57 | 39 |
|  | 122<br>58 | 250<br>186 | 186<br>58 | 3A |
|  | 123<br>59 | 251<br>187 | 187<br>59 | 3B |
|  | 124<br>60 | 252<br>188 | 188<br>60 | 3C |
|  | 125<br>61 | 253<br>189 | 189<br>61 | 3D |
|  | 126<br>62 | 254<br>190 | 190<br>62 | 3E |
|  | 127<br>63 | 255<br>191 | 191<br>63 | 3F |

|                                                                                       |           |    |
|---------------------------------------------------------------------------------------|-----------|----|
|  | 148<br>20 | 14 |
|  | 131<br>3  | 33 |

|                                                                                     |             |
|-------------------------------------------------------------------------------------|-------------|
|  | 105 233 169 |
|-------------------------------------------------------------------------------------|-------------|

|                                                                                     |             |
|-------------------------------------------------------------------------------------|-------------|
|  | 122 250 186 |
|-------------------------------------------------------------------------------------|-------------|

|                                                                                     |            |
|-------------------------------------------------------------------------------------|------------|
|  | 94 222 222 |
|-------------------------------------------------------------------------------------|------------|

|                                                                                       |            |
|---------------------------------------------------------------------------------------|------------|
|  | 95 223 223 |
|---------------------------------------------------------------------------------------|------------|



## Printer Tabbing

When using TAB to print on the screen, PET looks at the current position of the cursor first ( POS(0) ). If the TAB argument is less than the cursor's position on the line then the data is simply printed in the spaces immediately following the last character printed. If the argument is greater than or equal to POS(0), PET subtracts POS(0) from the argument and prints the resulting number of cursor-rights.

However, when printing to the printer, the cursor is usually in column zero and TAB acts like the SPC function (the printer has no "internal cursor"). Therefore, to make TAB work on the printer, print the data to the screen first then to the printer. This can be done with duplicate PRINT and PRINT# statements or more efficiently with one "dynamic" PRINT# statement. For example:

```
10 REM OPEN OUTPUT FILES
 SCREEN & PRINTER
20 OPEN 3,3,1
30 OPEN 4,4,0
40 PRINT# 3+X,"ABCDEFGHJKLMNOPQRSTUVWXYZ";
50 X=1-X : IF X THEN 40
```

Line 50 toggles X from 1 to 0 thus repeating line 40 only twice. The semi-colon is important else the POS(0) goes back to zero. When a carriage return is required on the printer the following might be inserted between PRINT# and toggle statements:

```
45 IF X THEN PRINT#4,CHR$(13);
```

Dynamic PRINT# statements are only more efficient if the DATA being printed is within quotes. If variables are used, more bytes are probably saved by duplicating the output statements.

## More on Printer Output

L. D. Gardner of Leisure Books Ltd., St. Laurent Quebec, wrote in with:

Dear Sirs,

Our company uses 2 32k Commodore Pets with Centronics printers\*.

The printers are used to produce formal reports of a statistical nature, and we have found that the most difficult task is not in programming the calculations, but in printing the data in acceptable format.

If 2 decimal places are required, a value of "302" is not printed as "302.00", which makes the reports difficult to read. Also, a very small value such as ".001" is printed as "1E -03".

I have come up with a subroutine to handle this problem. Undoubtedly, it could be shortened, but for clarity is shown in the detailed version.

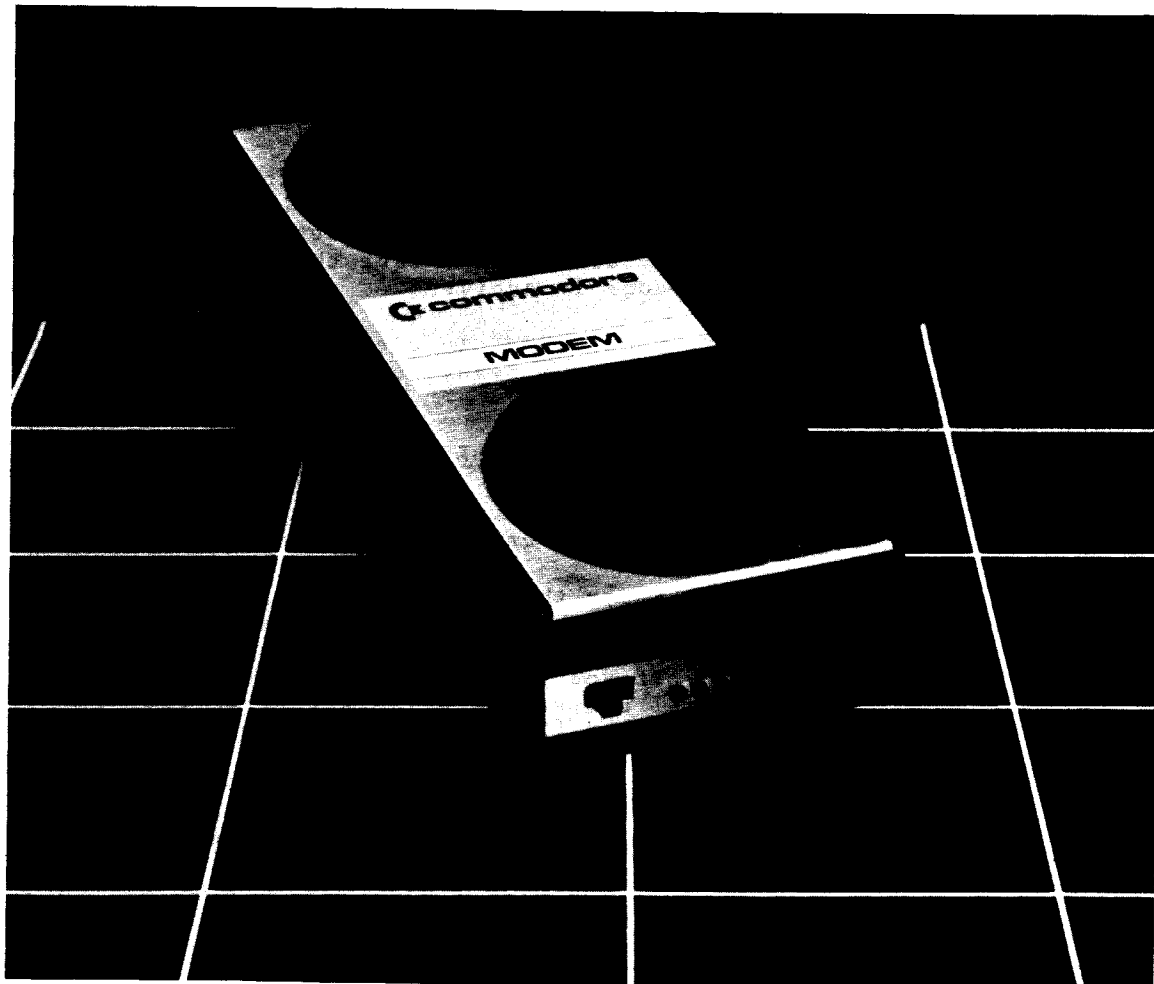
I am also enclosing a simplified program which illustrates how a program can be made to update itself. It has value when not sufficient data makes tape or disk files worthwhile. At the end of the program, DATA lines are printed and the cursor is moved up. All that's needed is to hit RETURN.

PROGRAM NAME: FORMAT 3DIG

```
10 REM *****
20 REM ***SUBROUTINE FOR TO FORMAT ***
30 REM ***NUMBERS TO 3 SIG. DIGITS ***
40 REM *****
100 Q$=".000":LQ=LEN(Q$):LT=LOG(10)
110 P=LQ-1:IF Q$="" THEN P=0
150 INPUT"NUMBER";A
160 A1=INT(A*10↑P+.5)/10↑P
165 IF A1=0 THEN 220
170 B1$=STR$(A1)
180 B2$=STR$(A1*10↑P)
190 LG=INT(LOG(ABS(A1)))/LT
195 IF ABS(A1)=1 THEN B$="1"+Q$
205 IF ABS(A1)<1 THEN B$=LEFT$(Q$,ABS(LG)+RIGHT$(B2$,LEN(B2$)-1))
215 IF ABS(A1)>1 THEN B$=MID$(B1$,2,LG+1)+". "+RIGHT$(B2$,P)
216 IF ABS(A)=1 THEN B$="1"+Q$
220 IF A1=0 THEN B$=Q$
230 IF A1<0 THEN B$="-"+B$
240 PRINT" ";A1,B$
250 GOTO150
```

SUBROUTINE TO FORMAT OUTPUT DATA

\* This would also apply to Commodore printers and others



CBM 8010 Modem

## Communications

As you are probably aware we are taking an active interest in communications hardware/software for the PET.

As communications is an area of special expertise Commodore is currently planning to initially release the following products through a small number of experienced communications dealers drawn from our current dealer network. These products will not be widely advertised at the present time but it was felt the readership should be kept informed of the latest happenings within Commodore.

Communicator 1, written by Cortex Computer Systems of Bedford is the latest product to be released and enables the PET to interface to DEC computers in particular.

The new 8032 with its 80 column screen, enables it to emulate a standard VDU terminal (ie. VT50), whilst in addition offering its local processing capabilities. An additional feature of Communicator 1 enables it to transfer files to and from

the host machine, giving it a distributed processing capability.

The days of a network of PETs communicating with each other may not be far away.

## Commodore Modem

Commodore have recently added a new peripheral to the current range, this is the 8010 Modem. It operates in Full or Half Duplex mode and it can both originate and answer.

The modem can be used for send/receive of disk files including Wordpro disk sequential files, disk spooling, PET networks and terminal emulation to name a few applications.

Communications between micro-computers is still virtually unknown territory to most users but it is a field which must grow in importance during the next few years if micro's are to gain their full potential. If you have communications experience then please write to the Editor with details of your achievements in this area.

# Machine Code

## THE 'UNWEDGE' -- A TAPE APPEND AND RENUMBER PROGRAM

Many PET users have only tape input for their system. (Until a few weeks ago, I was among that majority). If you have often wanted to append a favourite subroutine, you may have been stymied. I use the input subroutine from Cursor #4 ('INP', Oct 1978) in many programs. You may have your own favourites, if only there was an easy way to append them.

Another possibility might entail merging several short programs together, such as the Osborne & Associates: 'Some Common Basic Programs'. Individually, most are quite short, which could permit a 'menu-driven' composite program.

I am sure that other applications may come to mind.

Most of us have a Renumber program of some sort, whether it is in Machine Language, or as a Basic subroutine which is tacked on to the end. The main disadvantage is that the location in memory is fixed.

Welcome to the Upgrade-ROM combination of the two, which offers:

\*\*\*Machine Language for speed of operation.

\*\*\*Full relocatability to anywhere in RAM. When run, 'APPEND/RENUM' places itself in high memory and protects itself from intrusion by Basic.

\*\*\*The 'Unwedge' (or '<' key) attaches itself to the operating system. It may be used in the Direct Mode with a minimum of user input.

\*\*\*It is compatible with the Disk Wedge routine. Although the Append function works only with tape input, both Wedges may be active simultaneously.

\*\*\*The routine may be implemented and de-activated with the same 'SYS' command.

\*\*\*Uses only 771 bytes of memory.

Bill Seiler deserves the credit for nearly all the coding. In Pet User Notes (Vol 1, #7 Nov/Dec 1978), he presented his 'PET Renumber 3.0' for Original ROM. This has been converted to Upgrade ROM and allows for user input of parameters.

In our own Transactor (Vol 2, #3 July 1979), Bill presented the Append Wedge for Original ROM. In the same Pet User Notes issue, Jim Russo and Henry Chow gave us M7171. This is a high-monitor with merge capability for Original ROM. Features of both of these were converted to Upgrade ROM, and several additions made.

### TO USE THE PROGRAM:

1. Copy the program listing.

2. Save it immediately, since Machine Language routines have a nasty habit of crashing, with even the slightest error.

3. 'RUN' the program, and copy down the information which appears (after about 20 seconds).

4. Save the Machine Language code with the ML Monitor if you wish to use it in the same memory location every time.

5. Activate the routine with the given 'SYS' command. Since this has been made reversible, you may de-activate with the same call. (Don't save the program when the Unwedge is active).

6. Enter 'NEW' to clear the Basic portion. You are now ready to use 'APPEND/RENUM'.

### TO RENUMBER A BASIC PROGRAM IN MEMORY:

1. Clear the screen, and move the cursor 3/4 of the way down the screen.

2. Enter the '<' key in the first column of a line, then hit the 'R' key.

3. If you hit 'RETURN' now, the program will be renumbered, starting at 100, in steps of 10. (The default conditions).

4. For a different starting point, simply enter that number next. Be careful that the final line number will not exceed 63999.

5. If you wish a different step size, enter a comma (','), then the new increment. Values up to 255 are acceptable.

6. After you have hit 'RETURN', the message 'RENUMBERING' will be printed. The screen memory is used for storing the line numbers, so it will now show a variety of characters.

7. When the cursor returns (5-20 seconds), the job is complete. All 'GOTO', 'THEN', 'GOSUB' and 'RUN' destinations will be changed. Any illegal or unreferenced line numbers will be numbered '65535' as a flag to the error.

8. The screen memory will handle only about 500 lines. This shouldn't be a real limitation on most programs.

### TO APPEND ONE BASIC PROGRAM TO ANOTHER:

1. Place the program to be appended in Cassette #1 drive. The first line number of this program must be larger than the last one currently in memory. Use the Renumber function to prepare the program tape, if necessary. (This is the benefit of combining these two utilities).

2. Enter the '<' key in the first column of a screen line, then the letter 'A'. (You may enter the whole word, 'APPEND', but the program only checks the first letter).

3. If you wish a specific file to be Appended from tape, enclose the file name in quotes, just as with a normal Load.

4. If there is Machine Language after the end of Basic (of program in memory), then the routine aborts with message: '?NOT ALL BASIC PROGRAM ERROR'.

5. If the program on tape is Machine Language, then this program simply Loads normally, but does not Append. The same message is printed, but without the 'ERROR'. Be careful here because the the Basic pointers will have been changed.

6. If the whole program will not fit in memory, the routine aborts. An 'OUT OF MEMORY ERROR' message will be printed.

7. The message 'APPENDING' will be printed, while the routine searches for the correct program.

8. The routine accounts for the differences between Original and Upgrade ROM. The Original ROM saved an extra byte after the end of Basic.

I hope that this proves useful for you. I am particularly indebted to Jim Butterfield, for his counsel and patience through many elementary questions.

David A. Hook  
58 Steel Street  
BARRIE, Ontario L4M 2E9

PROGRAM NAME: APPEND/RENUM

```

10 EA=PEEK(53)*256+PEEK(52):B=770:SA=EA
 -B:J%=SA/256:J=SA-J%*256
20 POKEEA-1,J%:POKE53,J%:POKE49,J%:
 POKEEA-2,J%:POKE52,J%:POKE48,J
30 J%=(SA+31)/256:J=SA+31-J%*256:POKEEA
 -3,J%:POKEEA-4,J
40 FORI=SATOEA-5:READA$:A=VAL(A$)
50 IFLEFT$(A$,1)="H"THENJ=I+VAL(MID$(A$,
 2)):A=J/256:GOTO70
60 IFLEFT$(A$,1)="L"THENJ=I+VAL(MID$(A$,
 2)):A=J+.05-256*INT(J/256)
70 POKEI,A:NEXT
100 PRINT"ACTIVATE OR CANCEL APPEND/R
 ENUM":PRINTTAB(20)"--with: DATA
 A"
110 PRINT"SAVE WITH MLM":PRINT"ML$
 CHR$(34)"APPEND/RENUM"CHR$(34)",01"
120 X=SA/4096:GOSUB170:X=EA/4096:GOSUB1
 70:PRINT:PRINT:END
170 PRINT":":FORJ=1TO4:X%=X:X=(X-X%)*1
 6:IFX%>9THENX%=X%+7
180 PRINTCHR$(X%+48):NEXTJ:RETURN
10000 DATA173,L767,H766,133,52,173,L763
 ,H762,133,53,32,121,197,162,3,181
 ,120
10010 DATA72,189,L745,H744,149,120,104,
 157,L739,H738,202,208,241,96
10015 DATA201,60,208,8,72
10020 DATA 165,119,201,0,240,8,104,201,
 58,176,239,76,125,0,32,112
10030 DATA 0,201,65,240,7,201,82,208,23
 7,76,L282,H281,162,1,134,212
10040 DATA 202,134,209,134,157,169,2,13
 3,219,32,112,0,170,240,23,201
10050 DATA 34,208,246,166,119,232,134,2
 18,32,112,0,170,240,8,201,34
10060 DATA 240,4,230,209,208,242,32,86,
 246,32,18,248,32,10,244,165
10070 DATA 209,240,8,32,148,244,208,8,7
 6,110,245,32,166,245,240,248

```

```

10080 DATA 224,1,208,235,165,150,41,16,
 208,127,162,24,173,124,2,201
10090 DATA 4,240,7,162,0,32,L128,H127,2
 40,108,32,L123,H122,56,165,42
10100 DATA 233,2,133,42,165,43,233,0,13
 3,43,160,0,177,42,240,24
10110 DATA 32,L91,H90,32,110,242,169,13
 ,32,210,255,169,63,32,210,255
10120 DATA 162,1,32,L83,H82,76,119,195,
 32,L70,H69,177,42,208,3,32
10130 DATA163,H62,173,125,2,56,237,123,
 2,170,173,126,2,237,124,2
10140 DATA 168,165,42,56,233,2,141,123,
 2,165,43,233,0,141,124,2
10150 DATA 138,24,109,123,2,141,125,2,1
 52,109,124,2,141,126,2,197
10160 DATA 53,144,3,76,85,195,32,185,24
 3,76,221,243,32,L2,H1,230
10170 DATA 42,208,2,230,43,96,189,L11,H
 10,240,6,32,210,255,232,208
10180 DATA 245,96,13,78,79,84,32,65,76,
 76,32,66,65,83,73,67
10190 DATA 32,80,82,79,71,82,65,77,32,0
 ,13,65,80,80,69,78
10200 DATA 68,73,78,71,32,0,13,82,69,78
 ,85,77,66,69,82,73
10210 DATA 78,71,13,0,32,112,0,240,33,1
 76,249,32,115,200,72,166
10220 DATA 17,164,18,134,62,132,63,104,
 240,24,32,112,0,240,19,176
10230 DATA 249,32,115,200,166,17,134,66
 ,208,12,169,100,133,62,169,0
10240 DATA 133,63,169,10,133,66,162,36,
 32,L-115,H-116,169,254,133,33,169
10250 DATA 127,133,34,165,40,133,31,165
 ,41,133,32,32,L292,H291,160,3
10260 DATA 177,31,145,33,185,92,0,145,3
 1,136,192,1,208,242,177,31
10270 DATA 240,16,32,L301,H300,170,136,
 177,31,133,31,134,32,32,L267,H266
10280 DATA 208,220,169,255,200,145,33,2
 00,145,33,165,40,133,119,165,41
10290 DATA 133,120,208,3,32,L277,H276,3
 2,L274,H273,208,3,76,57,196,32
10300 DATA1266,H265,32,L263,H262,32,L26
 0,H259,170,240,233,162,4,221,L262
 ,H261
10310 DATA 240,5,202,208,248,240,238,16
 5,119,72,165,120,72,32,112,0
10320 DATA 176,230,32,115,200,32,L51,H5
 0,104,133,120,104,133,119,160,0
10330 DATA 162,0,189,1,1,240,15,72,32,1
 12,0,144,3,32,L82,H81
10340 DATA 104,145,119,232,208,236,32,1
 12,0,176,8,32,L102,H101,32,118
10350 DATA 0,144,248,201,44,240,192,208
 ,175,32,L134,H133,169,0,133,33
10360 DATA 169,128,133,34,160,1,177,33,
 197,18,240,21,201,255,208,24
10370 DATA 133,95,133,94,165,94,133,96,
 162,144,56,32,85,219,76,233
10380 DATA 220,136,177,33,197,17,240,23
 6,32,L96,H95,32,L116,H115,208,212
10390 DATA 32,L62,H61,160,0,177,31,200,
 145,31,32,L90,H89,208,8,230
10400 DATA 42,208,2,230,43,136,96,164,3
 1,208,2,198,32,198,31,76
10410 DATA1-29,H-30,32,L28,H27,160,1,17
 7,33,136,145,33,32,L56,H55,240
10420 DATA 5,32,L65,H64,208,239,164,42,
 208,2,198,43,198,42,96,165
10430 DATA 42,133,31,165,43,133,32,165,
 119,133,33,165,120,133,34,96

```

```

10440 DATA 165,62,133,94,165,63,133,95,
 96,165,94,24,101,66,133,94
10450 DATA 144,2,230,95,96,165,31,197,3
 3,208,4,165,32,197,34,96
10460 DATA 32,L2,H1,230,33,208,2,230,34
 ,96,160,0,230,119,208,2
10470 DATA 230,120,177,119,96,137,138,1
 41,167,76

```

## Input and Output from PET Machine Language

Jim Butterfield

Output to the screen is quite straightforward. Load the ASCII character into the A register; then call the routine at FFD2. Special characters, such as cursor movements, will be honoured in the usual way.

The GET activity gives no trouble, either, except for one minor situation. To do a GET, call FFE4 and the character available will appear in the A register. If you do not have a character available the subroutine will return zero in the A register. Since you can't get an ASCII zero from the keyboard, recognize this as a "no-character" situation and arrange to deal with it as desired.

INPUT is a little trickier. When you call FFCF for Input, you will get one character back. This first time you call, it will prompt and get an input, transferring it via the screen in the usual way; then it will edit out leading and trailing spaces and quote marks. After doing all this work, it will deliver the first character to you. On subsequent calls, it will deliver the following characters. When it has delivered the whole input, it will deliver a Return character to signal you have got it all. After that, it starts over.

Beginners will be happier using the GET call.

## Peripheral Input/Output

Surprisingly easy, once you have the above techniques mastered.

Start by OPENing the file in BASIC, before you go to machine language. When you are ready to begin the actual activity, the machine language sequence is as follows:-

```

Load X with the logical file number;
For INPUT or GET, call FFC6 to set the
input channel;
For output, call FFC9 to set the output
channel;

```

Now use you INPUT, GET or OUTPUT calls as described above.

Finally, restore the normal input/output channels with a call to FFCC. Careful! This routine changes the A register to zero.

Wind up your program in BASIC by closing all files, as usual.

When you are INPUTting or GETting from an external device, keep an eye on the status word, ST (located at 020C in original ROM, or at 96 in 2.0 ROM). It will warn you when you reach the end of an input file.

The above procedure is not too hard, and it is likely to carry through to newer versions of ROM when they appear.

## An Instring Utility for the 16/32K PET

Have you ever wanted to program something like .....

```

MID$ (AS, 10) = "Name, Address,
etc.....

```

..... well now you can!.....thanks to another fabulous routine by Bill Maclean of BMB CompuScience, Milton, Ontario. The routine works only with PETs using the 2.0 ROM set.

This is a little utility to allow a programmer to change a substring within a main string. Its primary uses are manipulating data records in disk files and setting up formatted printer or screen outputs. It is called with the following command:

```
SYS 826,A$,B$,X
```

This command string will cause the string A\$ to be placed within string B\$, starting at the 8th character. The A\$,B\$ and X are all variables. Any variables can be used. The programmer is responsible for assuring that the length of the main string is not exceeded.

The machine language routine can be entered using the resident monitor and cursor editing the screen display. The code is completely relocatable and can be placed anywhere or relocated anywhere. The calling address (826 above) should be the address of the first byte of the program.

PROG NAME: INSTRING.BAS

```

1 C$=" "+"IT IS A NICE DAY"
5 B$="NOT"
10 FORA=1TO30
20 A$=C$
30 SYS826,B$,A$,A:PRINTA$:NEXT
READY.

```

READY.

B\*

```

 PC IRQ SR AC XR YR SP
.: 0401 E62E 32 04 5E 00 F6
.:
.: 033A 20 F8 CD 20 9F CC A0 00
.: 0342 B1 44 85 00 C8 B1 44 85
.: 034A 01 C8 B1 44 85 02 20 F8
.: 0352 CD 20 9F CC A0 01 B1 44
.: 035A 85 0F C8 B1 44 85 10 20
.: 0362 F8 CD 20 9F CC 20 D2 D6
.: 036A A5 12 F0 03 4C 03 CE C6
.: 0372 11 A5 0F 18 65 11 85 0F
.: 037A 90 02 E6 10 A0 00 B1 01
.: 0382 91 0F C8 C4 00 D0 F7 60
.:
.: ?

```

READY.

Danny Doyle

DIMP: A machine language routine for the PET to handle algebraic input.

Listing 1 shows a small machine code program called DIMP (short for Direct Input Mode for Pet) which will enable both BASIC and machine language programs to process algebraic expressions. Note that the DIMP routine has been coded to run on the New ROMs.

Strangers to machine language programming will find listing 2 helpful. It shows a short BASIC program which will load DIMP into the second cassette buffer. Once loaded, DIMP can be accessed by SYS826. Listing 3 gives a simple example on how to use DIMP.

For those of you who understand 6502 machine code turn again to listing 1. As you can see the DIMP routine has only 16 instructions, so it can fit easily into the 2nd cassette buffer, or it can be relocated to any convenient memory space without modifications. For certain applications the code can be reduced to only 8 instructions - but more about that later. For the moment let us study listing 1 a little more closely and see what DIMP does. Put simply, when DIMP is called it performs much the same job as the BASIC interpreter when servicing a direct mode command from the keyboard. To see how this is done we will take a walk through the code. Let us assume that DIMP, as per the listing, has been loaded at the beginning of the 2nd cassette buffer at address \$033A. Somewhere in his BASIC program the user has coded a SYS826 command at the point where he wishes to input and execute an expression. When a SYS 826 command is executed control is passed to DIMP and this signals the start of phase 1.

The first thing that DIMP does (lines 1-4 in the listing) is to preserve an important pointer; a pointer which contains the address at which BASIC will resume scanning when DIMP has finished processing. DIMP then makes a call (line 5 of the listing) to a firmware routine which asks for a line of input from the keyboard and places the resulting code into the PET keyboard input buffer which is located at address \$0200. On return from the keyboard input routine the X and Y registers point to the start address (minus one) of the keyboard input buffer. Line 6 and 7 of the listing shows DIMP storing the X and Y register contents into the location in which they will be used by the interpreter's main scan routine in the next phase of processing. Phase 2 occurs in lines 8 & 9. Line 8 is a call to the scan routine and line 9 calls a ROM routine which converts any keywords in the input image into tokens. (All BASIC statements input into the PEare converted into a machine internal format made up of tokens. Tokens are simply a single byte representation of BASIC keywords such as:- READ, POKE, INPUT, DIM etc.). With the input image converted, DIMP enters its third and main phase.

After a second call to the scan routine (line 10) to reset the image pointer, the main task of evaluating the expression is performed (line 11) by a call to BASIC's statement execution routine. Note that if any syntax errors are present in the expression the BASIC program will abort and only a cold start using the RUN command will be possible. Assuming that the expression has been successfully evaluated, control is returned to DIMP at line 12. This marks the fourth and final phase of processing. All that remains to be done is to restore the statement pointer that was saved on entry and then return control to the BASIC program (lines 12-16).

Until now I have confined my description of DIMP to its ability to handle algebraic expressions entered at run-time. In fact, as well as handling algebraic expressions DIMP will also perform the following subset of BASIC commands:-  
 DIM; END; PEEK; POKE; PRINT; SAVE; STOP; SYS; WAIT; VERIFY.  
 Further, with some restrictions, DIMP will handle FOR...NEXT constructions such as:-  
 FOR I=1 TO N: PRINT SQR(N): NEXT  
 At the end of the loop the program will halt with a 'READY' message but this can be warm started by typing 'CONT'.

If you are interested in experimenting further with DIMP then here are a few things you might like to think about. Firstly, as shown in listing 1, DIMP has been coded to accept input from the keyboard; this need not be so. By replacing the call to the keyboard input routine in line 5 with a call to a routine of your own design some interesting things



can be done. For instance, a routine could be developed which would build an image into the keyboard input buffer, or some other area of memory and with the X and Y registers set appropriately it would then be possible to process a dynamically generated image. Alternatively, one could have 'canned' BASIC images within a machine language program. Then by setting up the X and Y registers with the address (minus one) of a particular image it would be processed by making a call to DIMP with line 5 deleted. Finally, I mentioned earlier that for some applications it would be possible to reduce the size of DIMP. Well if you are going to call DIMP from a machine language program only, you can delete lines 1-4 and 12-15. As already explained these lines handle the saving and restoring of the BASIC statement pointer, so they are not needed for machine language programs.

#### listing 1

| Line | Address | Code     | Instruction |
|------|---------|----------|-------------|
| 1    | 033A    | A5 77    | LDA \$77    |
| 2    | 033C    | 48       | PHA         |
| 3    | 033D    | A5 78    | LDA \$78    |
| 4    | 033F    | 48       | PHA         |
| 5    | 0340    | 20 6F C4 | JSR \$C46F  |
| 6    | 0343    | 86 77    | STX \$77    |
| 7    | 0345    | 84 78    | STY \$78    |
| 8    | 0347    | 20 70 00 | JSR \$0070  |
| 9    | 034A    | 20 95 C4 | JSR \$C495  |
| 10   | 034D    | 20 70 00 | JSR \$0070  |
| 11   | 0350    | 20 00 C7 | JSR \$C700  |
| 12   | 0353    | 68       | PLA         |
| 13   | 0354    | 85 78    | STA \$78    |
| 14   | 0356    | 68       | PLA         |
| 15   | 0357    | 85 77    | STA \$77    |
| 16   | 0359    | 60       | RTS         |

PROGRAM NAME: DIMP

```

10 DATA 165,119,072,165,120,072,032,111
20 DATA 196,134,119,132,120,032,112,000
30 DATA 032,149,196,032,112,000,032,000
40 DATA 199,104,133,120,104,133,119,096
50 REM -----
60 FOR A=826 TO 857
70 READ N
80 POKE A,N
90 NEXT
100 PRINT:PRINT"DIMP IS LOADED"
110 NEW
READY.

```

DIMP PROG2

```

10 PRINT"YES? ";
20 SYS826
30 IFA$<>"END"GOTO10
40 STOP
READY.

```

[DIMP is a solution looking for a cause, I have so far thought of two areas where it would be very useful but I would like to see what you, the readers can come up with. I am going to give 10 pounds worth of software to any applications of DIMP which get published in CPUCN. Ed]

Here is a very useful routines from Roger Davis, Chairman of Western Australia Commodore Computer Users Association

#### The 'Ultimate' Screen Save

Many ways have been seen regarding how to effectively save a screen display as a file suitable for retrieval later. Initially, I started by PEEKing every location on the screen, obtaining its value; then creating a sequential disk file of those values. On requiring that screen display all we have to do is open the file and POKE the data (starting at 32768) onto the screen. This is effective but takes quite a long time, and of course creates a large file (approx 14 blocks on the disk).

Next, I converted all the PEEK values into strings; this has the advantage of only requiring a small file (approx 5 blocks), and also will, when read print back much quicker because of the length of strings themselves. However, the conversion routine is complicated, takes time and I was sure there was a better way.

Whilst 'hacking around' I realised that if the screen is an area of memory, (which it is of course) there should be no reason why the screen display itself could not be saved as a program. To test this I simply jumped into the monitor with an 'SYS 1024' and then saved the screen display with:-

```
S"SCREEN",08,8000,83E7
```

The current screen display was saved as a program file called 'SCREEN'. When I loaded 'SCREEN' it automatically displayed the screen display at the time it was saved in much the same way as Dos Support 4.0 will display the disk directory without replacing any program in memory. Great; there was only one problem, this system had saved everything I had typed onto the screen and of course scrolled off the top four or five lines of the screen as well!! Nevertheless, the idea was worth pursuing, so I created the following machine code 'SAVE' routine which at any time during a program will, with a SYS call, save exactly the screen display. This program can be used in fact to save any area of memory, with a given file name to any device.

PROGRAM NAME: SCREEN.BAS

```

100 REM
110 REM*** SCREEN SAVE ROUTINE ***
120 REM BASIC SEGMENT.
130 REM BY ROGER DAVIS.
140 REM
150 REM
160 REM*** GET DRIVE NUMBER
170 REM
1000 INPUT"DRIVE NUMBER";D%: IF D%<>0
 AND D%<>1 GOTO1000
1010 OPEN15,8,15,"I"+STR$(D%): POKE871,
 D%
1100 REM
1110 REM*** GET FILE NAME
1120 REM
1130 INPUT"FILENAME";F$: L=LEN(F$): IF
 L=0 GOTO1130

```

```

1140 POKE844,L
1150 FORI=1TOL: A=ASC(MID$(F$,I,1)):
 POKE872+I,A: NEXT
1200 REM
1210 REM*** PROGRAM START AND END
1220 REM
1230 INPUT"START ADDRESS (32768)";S
1240 INPUT"END ADDRESS (33768)";E
1250 D=S:GOSUB1300:POKE848,R%:POKE852,H
 %
1260 D=E:GOSUB1300:POKE856,R%:POKE860,H
 %
1270 REM
1280 REM*** SAVE MEMORY
1290 REM
1295 END: SYS826: END
1300 REM
1302 REM*** GET HI AND LO
1305 REM
1310 H%=(D/256):R%=D-H%*256:RETURN

```

SCRSAVE.SRC.....PAGE 0001

| LINE# | LOC | CODE | LINE |
|-------|-----|------|------|
|-------|-----|------|------|

|      |      |          |                                   |
|------|------|----------|-----------------------------------|
| 0001 | 0000 |          | *****                             |
| 0002 | 0000 |          | *****                             |
| 0003 | 0000 |          | *****                             |
| 0004 | 0000 |          | *****                             |
| 0005 | 0000 |          | *****                             |
| 0006 | 0000 |          | *****                             |
| 0007 | 0000 |          | *****                             |
| 0008 | 0000 | BEGIN    | =\$033A                           |
| 0009 | 0000 | SAVE     | =\$F6AD                           |
| 0010 | 0000 | READY    | =\$C38B                           |
| 0011 | 0000 | DEVICE   | =\$08                             |
| 0012 | 0000 | LOSTAR   | =\$00                             |
| 0013 | 0000 | HISTAR   | =\$80                             |
| 0014 | 0000 | LOEND    | =\$E8                             |
| 0015 | 0000 | HIEND    | =\$83                             |
| 0016 | 0000 | SECON    | =\$D4                             |
| 0017 | 0000 | LONAME   | =\$67                             |
| 0018 | 0000 | HINAME   | =\$03                             |
| 0019 | 0000 | NAMLEN   | =\$09                             |
| 0020 | 0000 |          |                                   |
| 0021 | 0000 |          |                                   |
| 0022 | 033A |          | *=BEGIN                           |
| 0023 | 033A | A9 4C    | LDA #\$4C                         |
| 0024 | 033C | 8D FD 03 | STA \$03FD ; PROTECT?             |
| 0025 | 033F | A9 08    | LDA #DEVICE ; GIVE DEVICE NO.     |
| 0026 | 0341 | 85 D4    | STA SECON                         |
| 0027 | 0343 | A9 67    | LDA #LONAME ; LO ADDR OF FILENAME |
| 0028 | 0345 | 85 DA    | STA \$DA                          |
| 0029 | 0347 | A9 03    | LDA #HINAME                       |
| 0030 | 0349 | 85 DB    | STA \$DB                          |
| 0031 | 034B | A9 09    | LDA #NAMLEN ; LEN OF FILENAME     |
| 0032 | 034D | 85 D1    | STA \$D1                          |
| 0033 | 034F | A9 00    | LDA #LOSTAR ; LO START ADDRESS    |
| 0034 | 0351 | 85 FB    | STA \$FB                          |
| 0035 | 0353 | A9 80    | LDA #HISTAR                       |
| 0036 | 0355 | 85 FC    | STA \$FC                          |
| 0037 | 0357 | A9 E8    | LDA #LOEND ; LO END ADDRESS       |
| 0038 | 0359 | 85 C9    | STA \$C9                          |
| 0039 | 035B | A9 83    | LDA #HIEND                        |
| 0040 | 035D | 85 CA    | STA \$CA                          |
| 0041 | 035F | A2 00    | LDX #00                           |
| 0042 | 0361 | 20 AD F6 | JSR SAVE                          |
| 0043 | 0364 | 4C 8B C3 | JMP READY                         |
| 0044 | 0367 |          | .END                              |

This worked OK when tested, however, if a file created in this way is loaded from a BASIC program, the 'calling' program will RUN immediately after the new screen is displayed. Depending on the application required, this can be avoided as shown in the following (simple!) example:-

```
10 REM TEST 'SAVE' PROGRAM
20 STOP
30 LOAD"0:SCREEN",8
Type RUN30 to try this.
```

Another way of doing this is to change line 20 to "20 GOTO 40". There is no point in putting an END after the LOAD instruction as the program will not reach any line number after the LOAD.

The same system of operation can be used for a 'load' as shown in the enclosed program.

On a slightly different topic, with so many programs being written for Commodore in machine code or a combination of machine code and ASCII, it is sometimes difficult to find the start and end address of the program. This can be easily done in the following way:-

```
For disk:
1) enter'open8,8,8,"program
name",p,r'
For tape:
1)enter'open1'
2)enter'sys 1024'
3)enter'm 027a,0290'
```

The resulting memory display is as follows:-

```
027a = the type of file
027b-027c = starting address
027d-027e = ending address
027f- = file name.
```

Dave Middleton

Rogers solution to saving the screen as an area of memory using machine code prompted me to perform the same operation with BASIC. Lines 1010-1030 are the actual save routine. The rest of the program consists of examples of how it works and how to use it. The only fault with my version of the program is that the name of the file to be save can not be changed. There are two solutions to this:-

1. POKE the filename of the screen to be saved into the actual program.
2. SAVE the program under the name 'SCREEN' and then rename the file on disk.

Like DIMP screen save is a solution looking for a cause but I had great fun playing with it.

PROGRAM NAME: SCREENSAVER

```
10 IFPEEK(826)=0GOTO5000
15 OPEN1,8,15,"S1:SCREEN"
1000 REM **** SCREEN SAVER ****
1010 FORA=1TO4:POKE826+A,PEEK(39+A):
NEXT
1015 POKE40,0:POKE41,128:POKE42,0:POKE4
3,132
1020 SAVE"1:SCREEN",8
1030 POKE40,1:POKE41,4:POKE42,PEEK(829)
:POKE43,PEEK(830)
1040 PRINT"PRESS SPACE NOW AND ALSO WH
EN SCREEN HASLOADED":WAIT59410,4,4

2000 POKE826,0:LOAD"1:SCREEN",8
2005 REM *****
2010 PRINT"THIS NEVER GETS PRINTED":
GOTO2010
2020 REM *****
5000 WAIT59410,4,4:PRINT"END OF PROG":
POKE826,1
```

S. Donald Rossland, B.C.

#### MACHINE LANGUAGE CASE CONVERTER

This machine language program will convert strings to the correct upper/lower case condition for printing on CBM 2022/23 printers with an original ROM PET. It is relocatable so will operate anywhere in memory. The routine given here puts it in the second cassette buffer, but changing the location given in line 10100 will place it wherever you wish.

There are several things which must be done in order for the routine to operate correctly. These are best demonstrated by the following program.

```
0 ML$="" : GOSUB 10000
10 POKE 59468, 14
20 ML$="az123AZ"
30 PRINT ML$: OPEN 4,4 : PRINT#4,ML$
40 SYS 826
50 PRINT#4,ML$: CLOSE 4
60 PRINT ML$
70 LIST
10000 DATA 160, 2, 177, 124, 141, 251
10010 DATA 0, 200, 177, 124, 141, 252
10020 DATA 0, 200, 177, 124, 141, 253
10030 DATA 0, 172, 251, 0, 136, 177
10040 DATA 252, 201, 219, 176, 22, 201
10050 DATA 193, 144, 5, 56, 233, 128
10060 DATA 208, 11, 201, 65, 144, 9
10070 DATA 201, 91, 176, 5, 24, 105
10080 DATA 128, 145, 252, 192, 0, 208
10090 DATA 223, 96
10100 FOR A = 826 TO 881 : READ B
10110 POKE A, B : NEXT : RETURN
```

Note that line 20 is altered once the program is RUN. This is done by the SYS command in line 40.

Now alter line 20 to:

```
20 ML$ = ML$ + "az123AZ"
```

and reRUN from line 0. This time line 20 has not been changed in the listing. Whenever a string is formed by concatenation, the new string is stored in a location different from the original strings i.e. up in high RAM. It is this new location that has been altered. The major advantage in working on a string stored away from the program listing is that you don't have to worry if the string has been previously altered.

Now change line 0 to:

```
0 A = 0 : GOSUB 10000
```

and reRUN from line 0. Two points to note are:

1. Make sure that the variable string to be printed is #1 in the variable table, and
2. form the string to be printed by concatenation.

#### ASSEMBLY LANGUAGE LISTING OF UPPER/LOWER CASE CONVERTER

##### MOVE VARIABLE POINTERS TO ZERO PAGE

```
LDY 2 Set Y register offset.
LDA 124,Y Load A with byte from
 variable table pointed to
 by 124/125 + Y
STA 251,0 and move to location 251.
 This byte is the
 character count.
INY Increment offset.
LDA 124,Y
STA 252,0
INY Shift start address of
 string to zero page.
LDA 124,Y
STA 253,0
```

##### ADJUST STRING

```
LDY 251,0 Load Y with string
 character count from
 location 251
DEY Decrement Y offset.
 Y points to character to
 be altered next.
```

##### TEST FOR LOWER CASE

```
LDA 252,Y Load A with string byte
 pointed to by 252/253
 and offset by Y
CMP 219 and compare to lower
 case 'z' and
BCS 22 if greater than, skip to
 COMPARE Y.
CMP 193 Compare to lower case
 'a' and
BCC 5 if less than skip to TEST
 FOR UPPER CASE.
```

##### ADJUST LOWER CASE

```
SEC Set carry flag
SBC 128 Subtract 128 from the
 string byte in A and
 always skip to STORE
 MODIFIED CHARACTER.
```

##### TEST FOR UPPER CASE

```
CMP 91 Compare to upper case 'Z'
 and
BCS 9 skip to COMPARE Y if
 greater than.
CMP 65 Compare to upper case 'A'
 and
BCC 5 skip to COMPARE Y if less
 than.
```

##### ADJUST UPPER CASE

```
CLC Clear carry flag.
ADC 128 Add 128 to string byte.
```

##### STORE MODIFIED CHARACTER

```
STA 128,Y Store byte at location
 pointed to by 252/253
 and offset by Y.
```

##### COMPARE Y FOR STRING END

```
CPY 0 Compare Y to '0' and
BNE 223 skip to DEY in ADJUST
 STRING if string not
 finished.
```

```
RTS Otherwise, return to BASIC.
```

D.J.Pocock  
Training Department

#### INPUT : SHORT FORM

When writing BASIC programs with many different INPUT's many of you will have cursed. Because unlike most of the PET key words (PRINT etc) which have short forms of 2 or 3 characters (eg ? for PRINT) INPUT has no short form and all 5 characters have to be typed each time.

If you use '!' in place of INPUT when typing your program (eg 10 ! "WHAT IS YOUR NAME ";N\$), it is possible with a small (approx 100 Byte) machine code routine to convert every occurrence of '!' to INPUT.

The PET stores its key words as TOKENS in a single byte. For example INPUT is represented by 133(decimal) or \$85(hex). It is this method of storage which makes this routine simple.

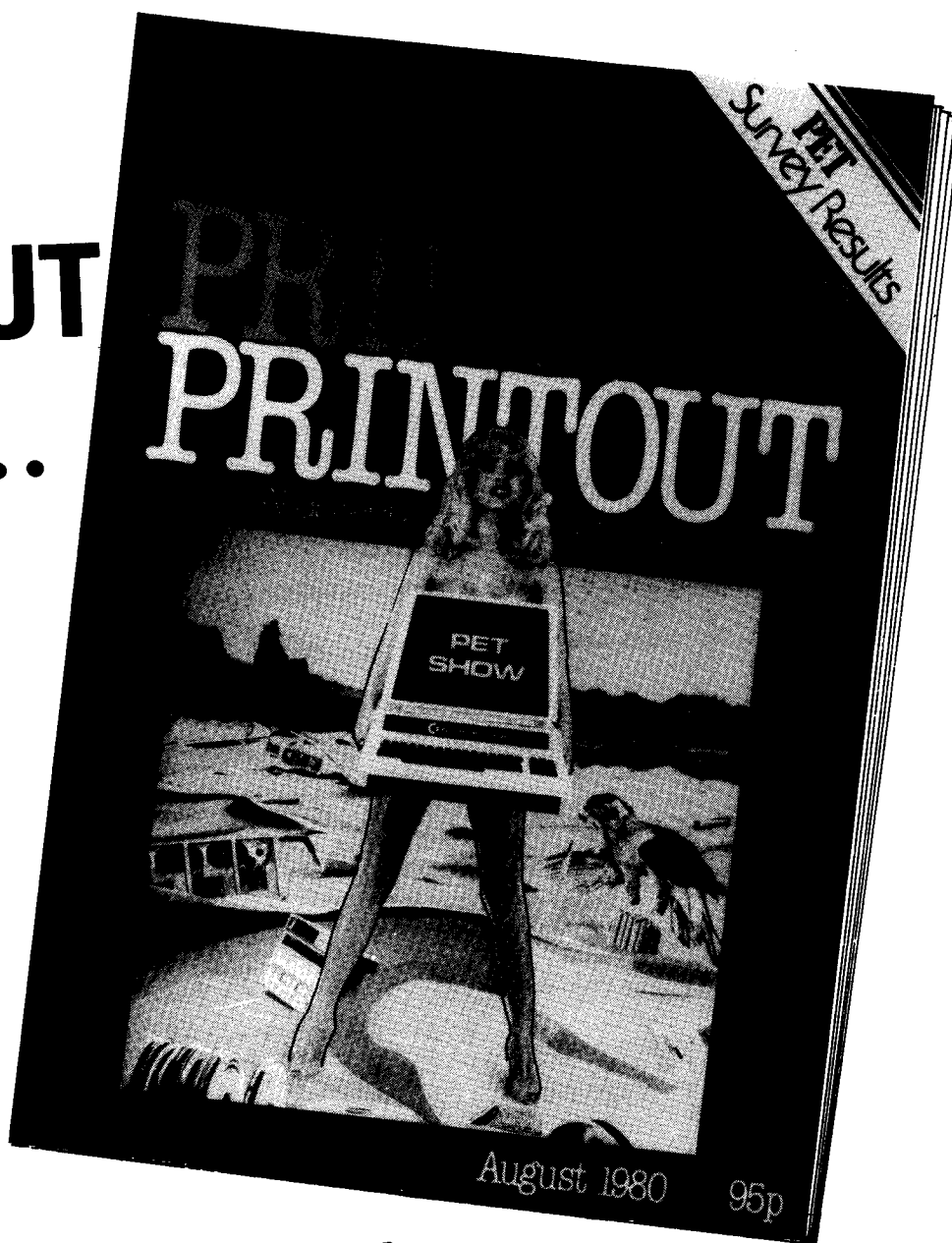
To be more specific the routine searches the BASIC text area for exclamation marks which are not enclosed in quotes, and converts these into the single character token for INPUT.

The routine can easily be altered to use other symbols (@ # & etc) to represent other basic key words. To alter the routine :-

POKE 943 with the ASC of the symbol

and POKE 944 with the value of the token (see THE PET REVEALED for a list of these).

# ALL ABOUT PET...



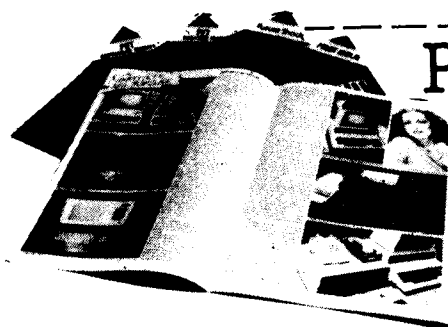
TEN TIMES A YEAR **PRINTOUT** PUBLISHES THE LATEST NEWS  
FROM THE FAST-MOVING WORLD OF THE CBM/PET.

In it you will find authoritative reports on the latest hardware, unbiased program reviews and a wealth of programming hints and listings. Whether you are a complete beginner or a hardened enthusiast, PRINTOUT is for you. Try a copy of the latest issue and see for yourself.

PRINTOUT's hardware and software reviews could save you a small fortune—we never pull our punches, especially about business software. PRINTOUT also tells you about the latest peripherals—speech synthesisers, voice recognition, high resolution graphics, even colour!

PRINTOUT can help solve your programming problems too; each issue we publish pages of readers questions with answers by our resident panel of PET experts. PRINTOUT is professionally produced, lavishly illustrated on glossy paper and packed with information. Last issue ran to 40 pages.

PET is the trademark of Commodore      Subscriptions are for a full volume only and start with the first issue of that volume.



**PRINTOUT** Greenacre, North Street, Theale, Berks RG7 5EX

I enclose: ☐ 95p for the latest issue      ☐ £9.50 for one years subscription (UK only)  
☐ £14.50 for one years subscription (o'seas)      ☐ £10.50 for one years subscription (Eire)

My name is *P.R.* .....

My address is .....

..... Post code .....

The routine sits in the second cassette buffer and is called by SYS 826. Below are two methods for loading the routine, source code and BASIC loader.

SOUR\*.....PAGE 0001

LINE# LOC CODE LINE

```

0001 0000 PTR = $00
0002 0000 *=$033A
0003 033A A9 00 LDA #$00
0004 033C 85 00 STA PTR ;LO
0005 033E A9 04 LDA #$04
0006 0340 85 01 STA PTR+1 ;HI BYTE START OF SEARCH
0007 0342 A0 00 LDY #$00 ;ZERO INDIRECT POINTER
0008 0344 20 99 03 ENDL JSR INCR
0009 0347 B1 00 LDA (PTR),Y ;EXAMINE LINKS
0010 0349 8D AC 03 STA EP ;FOR END OF
0011 034C 20 99 03 JSR INCR ;PROGRAM
0012 034F B1 00 LDA (PTR),Y ;
0013 0351 0D AC 03 ORA EP ;
0014 0354 F0 37 BEQ ENDP ;TOTAL LINK = 0
0015 0356 20 99 03 JSR INCR ;LINE # LO
0016 0359 20 99 03 JSR INCR ;LINE # HI
0017 035C A9 00 LDA #$00
0018 035E 8D AD 03 STA QTF ;CLEAR QUOTE FLAG
0019 0361 8D AE 03 STA REMF ;CLEAR REM FLAG
0020 0364 20 99 03 NCR JSR INCR ;FIRST / NEXT CHAR
0021 0367 B1 00 LDA (PTR),Y
0022 0369 F0 D9 BEQ ENDL ;END OF BASIC LINE
0023 036B C9 22 CMP #' ' ;QUOTES ?
0024 036D F0 1F BEQ QTMF ;YES FLIP QUOTE FLAG
0025 036F C9 8F CMP #$8F ;IS IT REM
0026 0371 D0 03 BNE CHAR ;NO
0027 0373 8D AE 03 STA REMF ;YES SET REM FLAG
0028 0376 CD AF 03 CHAR CMP SYMBL ;IS IT SYMBOL
0029 0379 D0 E9 BNE NCR ;NO NEXT CHARACTER
0030 037B AD AD 03 LDA QTF ;YES CHECK QUOTE FLAG
0031 037E D0 E4 BNE NCR ;QUOTE FLAG SET
0032 0380 AD AE 03 LDA REMF
0033 0383 D0 DF BNE NCR
0034 0385 AD B0 03 LDA TOKEN ;LOAD KEY TOKEN
0035 0388 91 00 STA (PTR),Y ;STORE IN BASIC TEXT
0036 038A 4C 64 03 JMP NCR ;NEXT CHARACTER
0037 038D 60 ENDP RTS ;FINISHED TO BASIC
0038 038E AD AD 03 QTMF LDA QTF
0039 0391 49 01 EOR #$01 ;FLIP QUOTE FLAG
0040 0393 8D AD 03 STA QTF
0041 0396 4C 64 03 JMP NCR
0042 0399 18 INCR CLC ;INCREMENT POINTER
0043 039A A5 00 LDA PTR
0044 039C 69 01 ADC #$01
0045 039E 85 00 STA PTR
0046 03A0 A5 01 LDA PTR+1
0047 03A2 69 00 ADC #$00
0048 03A4 85 01 STA PTR+1
0049 03A6 C9 80 CMP #$80
0050 03A8 F0 01 BEQ ERROR
0051 03AA 60 RTS
0052 03AB 00 ERROR BRK
0053 03AC 00 EP .BYTE 0
0054 03AD 00 QTF .BYTE 0
0055 03AE 00 REMF .BYTE 0
0056 03AF 21 SYMBL .BYTE '/'
0057 03B0 85 TOKEN .BYTE $85
0058 03B1 .END

```

ERRORS = 0000



```

10 REM ! TO GIVE INPUT
20 FOR L = 826 TO 944
30 READ C : POKE L , C
40 NEXT L
50 NEW
100 DATA169,0,133,0,169,4,133,1,160,0
110 DATA32,153,3,177,0,141,172,3,32,153
120 DATA3,177,0,13,172,3,240,55,32,153
130 DATA3,32,153,3,169,0,141,173,3,141
140 DATA174,3,32,153,3,177,0,240,217
150 DATA201,34,240,31,201,143,208,3,141
160 DATA174,3,205,175,3,208,233,173,173
170 DATA3,208,228,173,174,3,208,223,173
180 DATA176,3,145,0,76,100,3,96,173,173
190 DATA3,73,1,141,173,3,76,100,3,24
200 DATA165,0,105,1,133,0,165,1,105,0
210 DATA133,1,201,128,240,1,96,0,0,0,0
220 DATA33,133

```

```

10 REM ** DEMONSTRATION / TEST PROGRAM
**
20 SYS 826 : REM ENSURE ALL !'S ARE INP
UT FOR THIS RUN
30 REM THIS IS A TEST !!!
40 ! "WHAT IS YOUR NAME ";N$
50 ! "HOW OLD ARE YOU ";A
60 PRINT N$;" YOU ARE ABOUT";A*365;"DAY
S OLD !!!"
70 GOTO 30

```

## User Club & Bits and Pieces

### Dave Middleton Software Prizes

In Volume 2 Number 2, Andrew Goltz announced the introduction of the software prize for the best article in CPUCN, the prizes being 50 pounds for the best article in the issue and a 250 pound prize for the best article in the volume. Very little publicity has been given to this and I would suspect that most readers forgot all about it. The decision for who would get the prizes was fairly difficult. Jim Butterfield has been so prolific the he should by rights win all the prizes but that would be a bit pointless as he has access to any software he requires. Also a lot of material is produced 'in house', Mike Gross-Niklaus and Paul Higginbottom are prime examples.

This leaves a fairly small amount of original work produced outside Commodore UK. We have published a lot of small items about programming but there have been very few articles sent in which extend over more than one page but those we have had are generally very good.

Here are the 50 pound software prizes:

| No. | Name      | Article                          |
|-----|-----------|----------------------------------|
| 1   | D. Muir   | Digital to Analogue Conversion.  |
| 2   | Mike Todd | Owners Report.                   |
| 4   | RJ Leman  | Assembling an Assembler.         |
| 5   | LJ Slow   | Sorting by Insertion & chaining. |
| 7   | AR Clarke | Far infra-red astronomy          |
|     | CD Smith  | ground station.                  |
| 7   | DA Hills  | Supermon Old ROM.                |
| 8   | D Doyle   | DIMP.                            |
| 8   | R Davis   | The 'ultimate' screen save.      |

The 250 pound software prize for volume 2 goes to:

5 Bob Sparks TVA meter for the physics lab.

If the above people would like to write to me giving their choice of software from the current Petpack Master Library catalogue I will arrange to have the programs sent.

I hope this will tempt a few more of you into taking up the pen or better still Wordpro and putting your thoughts, applications, programs or ideas onto paper, it is suprisingly easy once you get started.

Dave Middleton

### A review of Adventure

Adventure is a very difficult game to describe because it is so unusual and because if too much information is given away then the game is spoilt.

First a very short history of the game. It was written by two programmers Willie Crowther and Don Woods as the Massachusetts Institute of Technology as an exercise in Artificial Intelligence. The original program was written in FORTRAN which is quite amazing as FORTRAN is not exactly renowned for its string handling capabilities! The game has had an enormous secret following for years as just about every main-frame computer has a copy lurking in the depths of its disk packs which is played by a small band of devotees at enormous cost to the company.

It can hardly be a suprise that PET Adventure has been made available by Jim Butterfield who has converted the FORTRAN code into its BASIC equivalent. The main Adventure program is 12k long and when the text files are included this comes to a massive 56k! To run Adventure you will need a 32k PET and a disk system with Adventure running on drive 0.

The program is text oriented which in this case is far better than trying to use graphics as a few well chosen words can build an imaginative picture that would take Walt Disney a month to put on film. I cannot describe any of the events which occur but maybe the following will give you the flavour of the game....

'Somewhere nearby is a colossal cave, where others have found fortunes in treasure and gold, though it is rumoured that some who enter are never seen again. Magic is said to work in the caves. I will be your eyes and hands.....'

The game is organised around a complex cave system with various challenges and objects hidden away within it. As the caves are explored and the treasure removed points are given with a maximum score of 300 being possible. I have never met anybody who has achieved this by themselves without either looking at the source code or taking clues from friends. How do I rate as an Adventurer? So far I have only scored 64 points!

If any User Club would like a copy of Adventure then please contact me at Commodore Slough with details of your club activities and I will send it to you. If there is no group in your area and you are willing to set one up then please contact me. Please note that I am unable to send out copies of Adventure to individual members at this time.

#### User groups

The following people are contacts for those of you who may be interested in joining IPUG (Independent PET User Group). Some of the groups are well established and others have only been formed recently, I would suggest that if you wish to meet other PET users for a chat or maybe a bit of assistance with programming problems you contact either the nearest group to you or Eli Pamphlett at the address given below for more details.

General Secretary:  
Eli Pamphlett  
The Coppers  
Sudbury Road  
Yoxall  
Burton Upon Trent  
Staffs  
Tel. Yoxall (0543) 472222

Mrs J.I Rich  
Central Electricity Gen. Board  
Research and Development Dept.  
Berkeley Nuclear Lab.  
Berkley  
Glos.

G.A Parkin  
Robert May's School  
West Street  
Odiham  
Hants

Maurice D Meredith  
Lect. Comp & Info Studies  
Dept of Education  
The University  
Southampton

Wg. Cdr. M.A.F Ryan  
164 Chesterfield Drive  
Riverhead  
Sevenoaks  
Kent

Mr Franklin  
Petalect Ltd.  
33/35 Portugal Road  
Woking  
Surrey

Paul Handover  
32 Long Wyre Street  
Colchester  
Essex

Brian Broomfield  
Little Orchard  
Hill Farm  
Radlett  
Herts

F.J Townsend  
The Mill  
Rhydowen  
Llandyssul  
Dyfed

M.J Merriman  
"Pippins"  
The Ridgeway  
Stourport on Severn

Raymond N. Davies  
105 Normanton Rd  
Derby

David W Jowett  
197 Victoria Road East  
Thorton-Cleveleys  
Blackpool

Eliot Khabie-Zeitoune  
9 Hamlea Close  
London SE12

Geoff Squibb  
108 Teddington Pk Rd  
Teddington  
Middex

Jim Cocallis  
20 Worcester Rd  
Newton Hall Estate  
Durham

Trevor Langford  
Allen Computers  
16 Hainton Avenue  
Grimsby  
South Humberside

Dr J MacBrayne  
27 Paidmyre Crescent  
Newton Mearns  
Glasgow

Geoff is interested in setting up a user group in the Slough area, if anybody is interested then please contact him at the address below.

Geoff Luxford  
51 Station Rd  
Burnham  
Nr Slough

Tel Burnham 2601

The following groups are not affiliated to IPUG but have some of the largest group memberships outside the Official Commodore User Club.

North London Computer Club  
North London Poly.  
Holloway  
London N7

S.U.P.A  
(Southern Users of Pet Association)  
Mr H. Pilgrim  
143e Ditchling Rd  
Brighton

Kent and Sussex User Group

A. French  
Crawley Teacher Centre  
Ashdown Drive  
Crawley

or

A. Updown  
West Sussex Institute  
Bognor College  
Bognor Regis  
Sussex

If I have omitted any user group from the above list then I apologise in advance but the problem is lack of communication. To this end I am going to keep a page of CPUCN set aside for club news so that the independent clubs can announce meetings and events.